

INTERFACE DATA DISPLAYING METHOD AND DEVICE

TECHNICAL FIELD

[0001] The present application relates to data presenting technologies, and in particular, to an interface data presenting method and device.

5

BACKGROUND

[0002] An interface framework of a client application is usually written by native codes. The interface framework can remain unchanged after being set in advance based on service needs. The framework and style of each interface can be adapted to a corresponding service data structure. If the interface framework needs to be changed
10 based on a change of service, or a new framework interface needs to be added to add new service data, the client application usually needs to be recompiled, re-downloaded, and re-installed. If a user does not update the client application, service data changes or additions cannot be presented to the user.

SUMMARY

15 [0003] The present application provides an interface data presenting method and device, so as to dynamically update a client application's interface.

[0004] According to a first aspect of an implementation of the present application, an interface data presenting method running on a client application is provided, where the method includes the following steps: parsing, to obtain a virtual view node,
20 structured data that is of an interface template and that is delivered by a server, where the structured data of the interface template is generated by the server by parsing interface template data written in a general protocol markup language, and the interface template data includes a common tag for describing a widget; binding service data to the virtual view node; and calculating location information of the
25 widget based on the virtual view node that the service data is bound to, and rendering a view based on the location information of the widget.

[0005] According to a second aspect of an implementation of the present application, an interface data presenting method running on a server is provided, where the method includes the following steps: parsing, to obtain structured data, interface template data written in a general protocol markup language, where the interface template data includes a common tag for describing a widget; and delivering the structured data to a client application, so that the client application renders the widget into a view.

[0006] According to a third aspect of an implementation of the present application, an interface data presenting device located on a client application is provided, and includes: a parsing engine module, configured to parse, to obtain a virtual view node, structured data that is of an interface template and that is delivered by a server, where the structured data of the interface template is generated by the server by parsing interface template data written in a general protocol markup language, and the interface template data includes a common tag for describing a widget; a binding module, configured to bind service data to the virtual view node; and a layout engine module, configured to calculate location information of the widget based on the virtual view node that the service data is bound to; and a rendering engine module, configured to render a view based on the location information of the widget.

[0007] According to a fourth aspect of an implementation of the present application, an interface data presenting device located on a server is provided, and includes: a parsing engine module, configured to parse, to obtain structured data, interface template data written in a general protocol markup language, where the interface template data includes a common tag for describing a widget; and a communications module, configured to deliver the structured data to a client application, so that the client application renders the widget into a view.

[0008] The present application implements the interface framework of the client application by using the interface template technology instead of native code, and correspondingly provides a solution for the client application to display the service data by using the interface template. As such, not only the interface can be updated without updating the client application, but also the server can parse the interface template data to obtain the structured data, so that developers can design the interface template in the general markup language. Therefore, the requirement for the developers to learn proprietary protocols can be eliminated, to make the solutions of

the present application more universal.

BRIEF DESCRIPTION OF DRAWINGS

[0009] FIG. 1 is a flowchart illustrating an interface data presenting method, according to an implementation of the present application;

5 [0010] FIG. 2 is a diagram illustrating a timing sequence of information interaction in an application example, according to the present application;

[0011] FIG. 3 is an architectural diagram illustrating hardware of an interface data presenting device, according to an implementation of the present application;

10 [0012] FIG. 4 is a software logic block diagram illustrating an interface data presenting device, according to an implementation of the present application; and

[0013] FIG. 5 is a software logic block diagram illustrating an interface data presenting device, according to another implementation of the present application.

DESCRIPTION OF EMBODIMENTS

15 [0014] Example implementations are described in detail here, and are presented in the accompanying drawings. When the following description refers to the accompanying drawings, unless specified otherwise, same numbers in different accompanying drawings represent the same or similar elements. Implementations described in the following example implementations do not represent all implementations in accordance with the present application. On the contrary, they are
20 only examples of devices and methods that are described in the appended claims in detail and that are in accordance with some aspects of the present application.

[0015] The terms used in the present application are merely used for the purpose of illustrating implementations, and are not intended to limit the present application. The terms "a", "said" and "the" of singular forms used in the present application and
25 the appended claims are also intended to include plural forms, unless otherwise specified in the context. It should also be understood that, the term "and/or" used here indicates and includes any or all possible combinations of one or more associated listed items.

[0016] It should be understood that although the terms "first", "second", "third",
30 etc. can be used in the present application to describe various information, the

information is not limited by the terms. These terms are merely used to differentiate information of the same type. For example, without departing from the scope of the present application, first information can also be referred to as second information, and similarly the second information can also be referred to as the first information.

5 Depending on the context, for example, words "if" used here can be explained as "while" or "when" or "in response to determining".

[0017] In the present application, an interface template can be separated from an client application, to help dynamically update the interface template, and allow an application developer to write the interface template based on a general protocol markup language. The client application can be loaded on various smart devices such as a mobile terminal, a tablet, or a computer. The solutions is not limited to a specific type of operating system on a smart device. For example, the operating system can be an iOS system, an Android system, a Windows system, an OS X system, etc. The general protocol markup language can be the HyperText Markup Language (HTML).
10
15 Certainly, other languages specified in standard general protocols suitable to make an interface are not excluded.

[0018] The interface template in the present application can be an interface framework that includes layout relationship, display style, operation control logic of several common elements, but does not include service data. The interface template can be implemented by using HTML, Cascading Style Sheets (CSS), and Javascript (JS). The common element can include a common tag for interface template data in HTML source code. A widget is described by using the common tag, for example, a Body tag, a Div tag, a Title tag, a Label tag, and an Img tag. The common element can also include an element that presents a display style. The display style can be
20
25 implemented by using CSS. The layout is specified by using CSS for an interface written in HTML. Certainly, other methods for specifying the interface layout are not excluded. JS can be used to control display content, display effects, layout relationship adjustment of the common tags, and to process user's operations.

[0019] After an interface template written by an application developer is
30 configured on a server, the server converts interface template data into structured data of the interface template, and sends the structured data of the interface template to a client application. Referring to FIG. 1, after the client application receives the structured data, a work process can include the following steps:

[0020] S101. The client application parses the structured data of the interface

template sent by the server to obtain a virtual view node.

[0021] S102. Bind service data to the virtual view node.

[0022] S103. Calculate location information of a widget based on the virtual view node that the service data is bound to.

5 **[0023]** S104. Render a view based on the location information of the widget.

[0024] A work process of the server can include a process of converting the interface template data, a process of distributing the structured data of the interface template, and a process of exchanging the service data with the client application.

[0025] To resolve the problem of poor parsing performance of HTML, before the
10 server sends the interface template data to the client application, the server first parses the interface template data to obtain the structured data, and then sends the structured data to the client application, thereby the process of parsing the HTML by the client application can be avoided. Selected structured data can have a compact data structure that can directly express an object mode, such as a JSON format and an XML format.
15 The JSON format is used as an example. The HTML can be parsed to obtain a document object mode (DOM). A DOM node stores related information of an HTML common tag. The DOM node describes a relative location of a widget in the interface, described by using the common tag. Because a syntax tree of the JSON format is simpler than a syntax tree of the HTML, the client application can parse data in the
20 JSON format more efficiently than HTML data.

[0026] Each interface template has a unique interface identifier to distinguish between different interface templates. After the interface template data is parsed to obtain the structured data, the structured data is distributed by the server to the client application, so that the client application subsequently displays the service data by
25 using the structured data. The structured data of the interface template includes an interface identifier.

[0027] A process of distributing an updated interface template to the client application can vary with different service scenarios. For example, after the interface template is updated, the server can actively distribute, to the client application, new
30 structured data obtained by parsing the interface template, or can distribute the structured data to the client application as needed, etc.

[0028] A work process of the client application can include a process of parsing and storing the structured data of the interface template, a process of exchanging the service data with the server, a process of binding the service data to the interface

template, and a process of rendering a view.

[0029] After the structured data of the interface template is sent by the server to the client application, the client application parses the structured data of the interface template to generate tree-structure virtual view nodes for storage. Each virtual view node can store basic information of a DOM node in the HTML, such as related information of the common tag, and can further include type information in the corresponding client application, information about a response event in the client application, etc. The location information of the widget described by using the HTML common tag can be calculated by using the virtual view node.

[0030] After a service data exchange procedure is triggered, the client application obtains the service data from the server, and the server can deliver a template identifier that corresponds to the service data to the client application when delivering the service data.

[0031] As an example, the process of binding the service data to the interface template can be implemented by using a JS script in a JS engine-driven interface template. The service data is processed by the JS engine. The JS engine executes the JS script to insert the service data into a corresponding virtual view node. For example, reference can be made to the following process: 1. If attributes and methods in native code (referred to as Native hereafter) of a device operating system (for example, an iOS system or an Android system) do not exist in the JS of the interface template, an interface of the JS engine can be extended, to bind the Native attributes and methods to the JS engine. The extended interface of the JS engine can include, for example, a JS entry function for receiving and processing the service data, or an interface of a rendering engine for calling Native. 2. The JS engine transmits the service data to the JS entry function. 3. The service data is inserted into the corresponding virtual view node by using the JS entry function.

[0032] In an example, the rendering engine can be instructed, by using the JS engine and the interface of the rendering engine for calling Native, to trigger the process of rendering the view. In the process, the client application can render the view by using a plurality of methods. For example, after location information of each widget is calculated, the widget can be rendered in a corresponding location of the view. In this case, an interface presenting effect is that widgets are presented to the user one by one. In another example, after location information of widgets in an entire view is calculated in a memory, the location information of the widgets can be fetched

from the memory, the widgets are rendered together into the view, and the complete view is presented to the user, so that the time of view rendering can be greatly reduced.

[0033] There can be a process of updating the structured data of the interface template obtained by the client application from the server. Different implementation procedures for sending structured data of the updated interface template to the client application can be designed based on different service scenarios. For example, after updating the interface template, the server can actively push the structured data to the client application. Alternatively, after triggering a service procedure, the client application sends a request to the server when the service data needs to be presented by using a new interface template. Therefore, as an example, a process of asking for structured data of the new interface template from the server can include: After obtaining the service data from the server, the client application can search for the interface template that corresponds to the service data based on an interface identifier in the service data; and if no interface template is identified, request the structured data of the interface template that corresponds to the interface identifier from the server.

[0034] After receiving the updated structured data, the client application parses the structured data to obtain a virtual view node, and updates the related information stored in the virtual view node. The virtual view node is bound to the service data based on the updated information, and the virtual view node that the service data is bound to is passed to the rendering engine by using the interface of the rendering engine for calling Native, to redraw the view.

[0035] The following is an example of implementing the solution in an application scenario. In the application example, based on different functions, client applications are classified into a service client application and an interface client application, and servers are classified into a service server and an interface server. The service client application and the interface client application can be integrated into and loaded to a same App on a mobile terminal. The interface server includes an HTML parsing engine and an interface template management module. The interface client application includes an interface-template structured data parsing engine, a JS engine, a layout engine, and a rendering engine.

[0036] Referring to FIG. 2, in step 201, the server preprocesses interface template data.

[0037] The server uses the HTML parsing engine to parse the interface template data to obtain structured data based on a JSON format. An HTML common tag in the interface template data, a CSS attribute associated with the common tag, and a JS script are parsed and then stored in a same JSON file of the interface template management module in a form of a DOM tree node. The interface template management module performs addition, deletion, or modification on these JSON files.

[0038] In step 202, the interface client application parses JSON data of the interface template.

[0039] After receiving the JSON data of the interface template, the interface client application uses the interface-template structured data parsing engine to parse the JSON data of the interface template to obtain the virtual view node. Information such as basic information of the DOM node of the interface template, type information in an application, and information about a response event in the application is stored in the virtual view node.

[0040] In step 203, the service client application exchanges service data with the service server.

[0041] The service client application sends a request to the service server to obtain the service data. The service server encapsulates the needed service data in a predetermined format and adds an ID of the interface template to send the service data in the form of JSON data to the service client application.

[0042] In step 204, the service data is bound to the virtual view node.

[0043] After receiving the service data, the service client application forwards the service data to the JS engine of the interface client application for processing. The JS engine of the interface client application searches for the JSON data of the corresponding interface template based on the interface template ID, and transmits the service data to a JS entry function of the interface template for processing. The JS entry function splits the service data and then inserts the service data into the corresponding virtual view node for binding. For example, service data related to a Body tag is bound to a virtual view node that stores the Body tag, and service data related to a Div tag is bound to a virtual view node that stores the Div tag.

[0044] In the present step, if no JSON data of a corresponding interface template is identified based on the interface template ID, the interface client application needs to request the JSON data of the interface template that corresponds to the interface template ID from the interface server again.

[0045] In step 205, the layout engine calculates a widget location.

[0046] The layout engine calculates a location of each widget by using information stored in each virtual view node, and stores the location information of each widget in a memory.

5 **[0047]** In step 206, a view is rendered.

[0048] In this example, after locations of widgets are calculated, when the view is rendered, the rendering engine is instructed, by using the JS engine, to perform view rendering. The view rendering engine obtains the calculated location of each widget in the entire view from the memory, and presents the complete view.

10 **[0049]** Corresponding to the implementation of the interface data presenting method, the present application further provides an implementation of an interface data presenting device.

[0050] The implementation of the interface data presenting device in the present application can be applied to a smart terminal and a server. The device implementation can be implemented by using software, or can be implemented by using hardware or a combination of software and hardware. Software implementation is used as an example. As a logical device, the device is formed by reading a corresponding computer program instruction in a non-volatile memory to a memory by a processor of the smart terminal or the server where the device is located. From a hardware perspective, as shown in FIG. 3, FIG. 3 is a structural diagram illustrating hardware where a data presenting device in the present application is located. In addition to processor, memory, network interface, and non-volatile memory shown in FIG. 3, the smart terminal or the server where the device in this implementation is located can include other hardware based on actual functions of the smart terminal and the server. Details are not described here to avoid duplication.

[0051] Referring to FIG. 4, an interface data presenting device 400 can be located on a client application in an implementation, and include a parsing engine module 401, a binding module 402, a layout engine module 403, and a rendering engine module 404.

30 **[0052]** The parsing engine module 401 can be configured to parse, to obtain a virtual view node, structured data of an interface template sent by a server. The structured data of the interface template is generated by the server by parsing interface template data written in a general protocol markup language. The interface template data includes a common tag for describing a widget.

[0053] The binding module 402 can be configured to bind service data to the virtual view node.

[0054] The layout engine module 403 can be configured to calculate location information of the widget based on the virtual view node that the service data is bound to.

[0055] The rendering engine module 404 can be configured to render a view based on the location information of the widget. The general protocol markup language can be HTML, and the structured data can be JSON data or XML data.

[0056] Information stored in the virtual view node can include related information of the common tag, and/or type of the client application, and/or information about a response event in the client application.

[0057] The parsing engine module 401 can be further configured to instruct the virtual view node to update the stored related information after the client application receives structured data of an updated interface template.

[0058] In addition, a service data module 405 can be included, and is configured to receive the service data delivered by the server and send the service data to the parsing engine module 401, and request the service data from the server based on a notification from the parsing engine module 401.

[0059] The parsing engine module 401 can be further configured to send a notification to the service data module 405 when no structured data of an interface template that corresponds to the service data is identified.

[0060] That the rendering engine module 404 renders a view based on the location information of the widget includes: after the layout engine module 403 calculates location information of widgets, obtaining the location information of the widgets, and rendering the widgets into the view.

[0061] FIG. 5 is an implementation of an interface data presenting device 500. The device 500 is located on a server side, and includes: a parsing engine module 501, configured to parse, to obtain structured data, interface template data written in a general protocol markup language, where the interface template data includes a common tag for describing a widget; and a communications module 502, configured to deliver the structured data to a client application, so that the client application renders the widget into a view.

[0062] The general protocol markup language can be HTML, the structured data can be JSON data, and the common tag can be stored in a DOM node for the JSON

data.

[0063] For an implementation process of functions and roles of each unit in the device, reference can be made to an implementation process of a corresponding step in the foregoing method. Details are not described here for simplicity.

5 **[0064]** The apparatus implementation can correspond to the method implementation, and therefore is described briefly for simplicity. For related parts, reference can be made to descriptions in the method implementation. The described apparatus implementation is merely an example. The units described as separate parts may or may not be physically separate, and parts displayed as units may or may not
10 be physical units, can be located in one position, or can be distributed on a plurality of network units. Part or all of the modules can be selected based on actual needs to achieve the objectives of the solutions of the present application. A person of ordinary skill in the art can understand and implement the implementations of the present application without creative efforts.

15 **[0065]** The foregoing descriptions are merely examples of the implementations of the present application, but are not intended to limit the present application. Any modification, equivalent replacement, or improvement made without departing from the spirit and principle of the present application shall fall within the protection scope of the present application.

CLAIMS

1. A method for rendering data at an application interface, the method comprising:
parsing (S101), by a client application, structured data of an interface template, wherein
the interface template is delivered by a server to the client application to obtain a document
5 object model node, wherein the structured data of the interface template is generated by the
server by parsing interface template data written in a general protocol markup language, and the
interface template data comprises a common tag for describing a widget, wherein the document
object model node stores information related to the common tag, and wherein the document
object model node comprises location information of the widget in the interface described by
10 using the common tag, and wherein the interface template comprises a first unique interface
identifier to distinguish between interface templates;

based on the parsed structured data, generating, by the client application, a virtual view
node as part of a tree-structure of virtual view nodes for storing information for document object
model nodes including related information to the common tag;

15 binding (S102), by the client application, service data to the virtual view node from the
tree-structure of the virtual view nodes, wherein the client application searches for an interface
template that corresponds to the service data based on an interface identifier in the service data
that corresponds to the first unique interface identifier;

calculating (S103) location information of the widget based on the stored information in
20 the virtual view node that the service data is bound to;

rendering (S104), by the client application, the widget in a view of an interface of the
client application at a corresponding location of the view based on the location information of
the widget, wherein the service data is inserted into the widget in the view (S104); and

triggering by the client application a service procedure for presenting the service data;

25 after obtaining the service data from the server, searching, by the client application, for an
interface template that corresponds to the service data based on an interface identifier in the
service data; and

if no interface template is identified, requesting, by the client application, updated
structured data of the new interface template that corresponds to the interface identifier from the
30 server.

2. The method according to claim 1, wherein the general protocol markup language is HTML, and the structured data is JSON data or XML data.

3. The method according to any one of claims 1 to 2, wherein the virtual view node stores information related to the common tag, and/or type of the client application, and/or a
5 response event in the client application.

4. The method according to any one of claims 1 to 3, wherein the method further comprises:

after receiving updated structured data of an updated interface template, updating, by the client application, the related information stored in the virtual view node.

10 5. The method according to claim 1, further comprising:

after receiving the updated structure data from the server, the client application, parsing the updated structure data to obtain an updated virtual view node;

updating the related information stored in the virtual view node, wherein the virtual view node is bound to the service data based on the updated information; and

15 passing the virtual view node to a rendering engine to re-render the virtual view node.

6. The method according to any one of claims 1 to 5, wherein rendering the widget in the virtual view node based on the location information of the widget comprises:

after the location information of widgets is calculated, obtaining the location information of the widgets, and rendering the widgets into the view.

20 7. The method according to any one of claims 1 to 6, further comprising:

delivering the structured data to the client application, so that the client application renders the widget into the virtual view node.

8. The method according to any one of claims 1 to 7, wherein the structured data is JSON data and the common tag is stored in the document object model node for the JSON data.

25 9. The method according to any one of claims 1 to 8, wherein the structured data comprises a first syntax tree with a format simpler than a second syntax tree of the interface

template.

10. The method according to any one of claims 1 to 9, wherein binding the service data to the virtual view node comprises inserting the service data in the virtual view node by using an entry function.

5 11. The method according to any one of claims 1 to 10, further comprising:
searching, by the client application, for the new interface template to correspond to the service data based on an interface identifier in the service data; and
if no interface template is identified based on the searching, requesting structure data of the interface template corresponding to the interface identified from the server.

10 12. An apparatus adapted to display interface data, the apparatus comprising a plurality of modules configured to perform the method of any one of claims 1 to 11.