

METHOD AND APPARATUS REUSING LISTCELL IN HYBRID APPLICATION

TECHNICAL FIELD

[0001] The present application relates to the computer field, and in particular, to a
5 method and an apparatus reusing a ListCell in a hybrid application.

BACKGROUND

[0002] Hybrid development is developing programs by using both front-end
development and native development. The front-end development is developing web
pages and web apps based on technologies such as JavaScript, HTML, and CSS. The
10 native development is programming based on native functions of local operating
systems of smart devices such as mobile phones. For example, native interface
development based on systems such as iOS and Android. React is a JS framework
produced by Facebook, and is a construction system for standardizing the front-end
development. React Native is launched by Facebook based on the React standards,
15 and is a technology implementation framework that native applications can be
developed by using front-end technologies.

[0003] The hybrid development is a very popular development mode for current
mobile APPs. In this development mode, a front-end web page development
technology is used to develop interfaces and services, and the native development is
20 used to provide native capabilities of various OS, so that an APP can have the
front-end dynamic capability and the native experience effect. A ListView is the most
common presentation form of an APP and it is used to present a page including a list
form content, presents a large number of ListCells with the same or similar structure
in a list, and usually can support inclusion of various ListCell. The ListCells have
25 basically the same presentation structure, and only information contents to be
presented are different. A ListCell is a content presented in each row or each sub cell
of the ListView. Because a screen has a limited display area, the screen can present
only some ListCell contents of the ListView at the same time. Some of the other

ListCells can scroll in the screen display area only by scrolling up and down, and correspondingly, some ListViews scroll out of the display area. Therefore, ListCells that a user can see on the screen are always limited. However, a front end needs to trigger Native to create instances and perform rendering for all ListCells in the current
5 front-end technology. Consequently, much performance is wasted. An existing ListView implementation solution on React Native is shown in FIG. 1. Because a ListCell is defined at a front-end layer, a React Native framework can only map an entire structure of a ListView to a native layer. Therefore, if the ListView has 100 ListCells, 100 ListCells are created at the native layer, and the 100 ListCells are
10 arranged and rendered, which consumes both memory and time.

[0004] In a list scrolling process, a ListCell that has been moved out of the displayable area can be reused to present a ListCell that is to be moved to the displayable area. As such, memory and time needed for creating and arranging new ListCells can be reduced to improve the performance. In the current hybrid
15 technology, in the ListView development, a structure layout and a ListCell class are defined at the front end, and the front end triggers the native layer to create a ListCell instance. Therefore, due to the limitation of a front-end DOM rendering mechanism, the native layer cannot efficiently reuse the ListCell because reconstructing a DOM needs higher costs.

[0005] In the current native development, there are corresponding technologies for reusing ListCells. However, because the front-end technology is used to develop pages in the hybrid technology, both a ListView and a ListCell are created at the front end. If the ListCell is created and reused at the native layer, a ListCell class defined at the front end cannot be transmitted to the native layer. In other words, an instance
20 cannot be directly created at the native layer for the ListCell class defined by using JavaScript (JS) at the front end. Therefore, currently, in the hybrid development, there is no technology that can be used to reuse a front-end ListCell in the industry.

SUMMARY

[0006] An objective of the present application is to provide a method and an
30 apparatus reusing a ListCell in a hybrid application, so as to resolve the problem that a structure and a class of a ListCell are defined at a front end but the ListCell is created and reused at a native end in the hybrid development.

[0007] According to an aspect of the present application, a method reusing a ListCell in a hybrid application is provided, where the method includes the following: defining a prototype of a ListCell at a front end of the hybrid application; and obtaining the ListCell through copying at a native end of the hybrid application based on the prototype of the ListCell, and reusing the ListCell.

[0008] Further, in the method, the defining a prototype of a ListCell includes the following: defining a structure layout and a class of the prototype of the ListCell.

[0009] Further, in the method, the prototype of the ListCell is a prototype of a single ListCell.

10 [0010] Further, in the method, the obtaining the ListCell through copying based on the prototype of the ListCell includes the following: creating a ListCell template based on the prototype of the ListCell; obtaining the ListCell by copying the ListCell template; and filling an obtained ListCell content in the copied ListCell, and displaying the ListCell.

15 [0011] Further, in the method, the obtaining the ListCell by copying the ListCell template includes the following: traversing and obtaining all sub-views of the ListCell template; and creating a ListCell that includes all the sub-views and attributes of the sub-views.

[0012] Further, in the method, the filling an obtained ListCell content in the copied ListCell includes the following: filling the obtained ListCell content in each corresponding sub-view of the created ListCell.

[0013] Further, in the method, the ListCell content is obtained from the front end.

[0014] Further, in the method, during obtaining the ListCell by copying the ListCell template, the number of copied ListCells is determined based on the number of ListCells that can be included and displayed in a display area.

25 [0015] Further, in the method, the reusing the ListCell includes the following: recycling a ListCell including the moved-out ListCell content as a reusable ListCell when a ListCell content is moved out of the display area; determining whether there is a reusable ListCell when a new ListCell content is to be moved to the display area; and if yes, filling the new ListCell content in the reusable ListCell, and displaying the ListCell; or if no, obtaining the ListCell by copying the ListCell template, filling the new ListCell content in the copied ListCell, and displaying the ListCell.

[0016] According to another aspect of the present application, an apparatus reusing a ListCell in a hybrid application is further provided, where the apparatus

includes: a definition apparatus, configured to define a prototype of a ListCell at a front end of the hybrid application; and a copy and reuse apparatus, configured to obtain the ListCell through copying at a native end of the hybrid application based on the prototype of the ListCell, and reuse the ListCell.

5 **[0017]** Further, in the apparatus, the definition apparatus is configured to define a structure layout and a class of the prototype of the ListCell.

[0018] Further, in the apparatus, the prototype of the ListCell is a prototype of a single ListCell.

10 **[0019]** Further, in the apparatus, the copy and reuse apparatus is configured to create a ListCell template based on the prototype of the ListCell, obtain the ListCell by copying the ListCell template, fill an obtained ListCell content in the copied ListCell, and display the ListCell.

15 **[0020]** Further, in the apparatus, the copy and reuse apparatus is configured to traverse and obtain all sub-views of the ListCell template, and create a ListCell that includes all the sub-views and attributes of the sub-views.

[0021] Further, in the apparatus, the copy and reuse apparatus is configured to fill the obtained ListCell content in each corresponding sub-view of the created ListCell.

[0022] Further, in the apparatus, the ListCell content is obtained from the front end.

20 **[0023]** Further, in the apparatus, the copy and reuse apparatus is configured to determine the number of copied ListCells based on the number of ListCells that can be included and displayed in a display area.

25 **[0024]** Further, in the apparatus, the copy and reuse apparatus is configured to recycle a ListCell including the moved-out ListCell content as a reusable ListCell when a ListCell content is moved out of the display area; determine whether there is a reusable ListCell when a new ListCell content is to be moved to the display area; and if yes, fill the new ListCell content in the reusable ListCell, and display the ListCell; or if no, obtain the ListCell by copying the ListCell template, fill the new ListCell content in the copied ListCell, and display the ListCell.

30 **[0025]** Compared with the existing technology, in the present application, the prototype of the ListCell is defined at the front end. The prototype of the ListCell is copied at the native end. The native end reuses an invisible ListCell based on a current list presentation location. As such, the prototype of the ListCell can be defined at the

front end, and the ListCell can be freely created, rendered, and reused at the native end. Therefore, not only the flexibility of defining the ListCell in the hybrid development can be reserved, but also the ListCell can be reused at the native end, thereby greatly improving the large-list presentation and operation performance. The core of the technical solutions of the present application is to resolve the problem that a structure and a class of a ListCell are defined at a front end but the ListCell is created and reused at a native end in the hybrid application development, so as to reuse the ListCell on a list page and improve rendering and operation performance.

BRIEF DESCRIPTION OF DRAWINGS

[0026] Other features, objectives, and advantages of the present application become more apparent by reading detailed description of the non-limitative implementations with reference to the following accompanying drawings.

[0027] FIG. 1 shows an implementation of an existing ListView;

[0028] FIG. 2 is a flowchart illustrating a method reusing a ListCell in a hybrid application, according to an aspect of the present application;

[0029] FIG. 3 is a principle diagram illustrating an example implementation of the present application;

[0030] FIG. 4 is a principle diagram illustrating another example implementation of the present application;

[0031] FIG. 5 is a principle diagram illustrating yet another example implementation of the present application; and

[0032] FIG. 6 is a diagram illustrating modules of an apparatus reusing a ListCell in a hybrid application, according to another aspect of the present application.

[0033] The same or similar reference numerals in the accompanying drawings represent the same or similar components.

DESCRIPTION OF IMPLEMENTATIONS

[0034] The following describes the implementations of the present application in detail with reference to the accompanying drawings.

[0035] In a typical configuration of the present application, a terminal device, a service network device, and a trusted party each include one or more central

processing units (CPU), an input/output interface, a network interface, and a memory.

[0036] The memory can include a non-persistent memory, a random access memory (RAM), and/or a nonvolatile memory in a computer readable medium, for example, a read-only memory (ROM) or a flash memory (flash RAM). The memory
5 is an example of the computer readable medium.

[0037] The computer readable medium includes persistent, non-persistent, movable, and unmovable media that can implement information storage by using any method or technology. Information can be a computer readable instruction, a data structure, a program module, or other data. An example of a computer storage
10 medium includes but is not limited to a phase change memory (PRAM), a static random access memory (SRAM), a dynamic random access memory (DRAM), a random access memory (RAM) of another type, a read-only memory (ROM), an electrically erasable programmable read-only memory (EEPROM), a flash memory or another memory technology, a compact disc read-only memory (CD-ROM), a digital
15 versatile disc (DVD) or other optical storage, a cassette magnetic tape, a tape and disk storage or another magnetic storage device, or any other non-transmission media. The computer storage medium can be configured to store information that can be accessed by a computing device. As described in the present specification, the computer readable medium does not include a transitory computer readable medium such as a
20 modulated data signal and a carrier.

[0038] As shown in FIG. 2, according to an aspect of the present application, a method reusing a ListCell in a hybrid application is provided. The method includes the following steps.

[0039] Step S1: Define a prototype of a ListCell at a front end of the hybrid
25 application.

[0040] Step S2: Obtain the ListCell through copying at a native end of the hybrid application based on the prototype of the ListCell, and reuse the ListCell. In the present application, the prototype (a parent) of the ListCell is defined at the front end. The prototype of the ListCell is copied (cloned) at the native end (Native). The native
30 end reuses an invisible ListCell based on a current list presentation location. As such, the prototype of the ListCell can be defined at the front end, and the ListCell can be freely created, rendered, and reused at the native end. Therefore, not only the flexibility of defining the ListCell in the hybrid development can be reserved, but also the ListCell can be reused at the native end, thereby greatly improving the large-list

presentation and operation performance. The core of the technical solution of the present application is to resolve the problem that a structure and a class of a ListCell are defined at a front end but the ListCell is created and reused at a native end in the hybrid development. As such, the ListCell can be reused on a list page to improve the rendering and operation performance. The present application is applicable to all hybrid APP developments, and has a very good implementation effect in a React Native framework.

[0041] In an example implementation of the method reusing a ListCell in a hybrid application in the present application, step S1 of defining a prototype of a ListCell includes the following: defining a structure layout and a class of the prototype of the ListCell. Here, defining a prototype of a ListCell includes defining a structure and a class of the ListCell. As such, the ListCell can be flexibly defined at the front end in the hybrid development.

[0042] In an example implementation of the method reusing a ListCell in a hybrid application in the present application, the prototype of the ListCell is a prototype of a single ListCell. Here, only one instance (that is, the prototype of the ListCell) is created at a front-end layer. Subsequently, a ListCell template can be created at the native end only based on the prototype. Then the ListCell template is copied as needed. As such, memory and time needed for creating and arranging new ListCells can be reduced to improve the performance.

[0043] In an example implementation of the method reusing a ListCell in a hybrid application in the present application, obtaining the ListCell through copying based on the prototype of the ListCell includes the following: creating a ListCell template based on the prototype of the ListCell; obtaining the ListCell by copying the ListCell template; and filling an obtained ListCell content in the copied ListCell, and displaying the ListCell. Here, as shown in FIG. 3, the ListCell template, that is, a ListCell prototype instance in the figure, is created at the native end only based on the prototype defined at the front end. Then, the ListCell template is copied based on the number of ListCell contents that need to be displayed to obtain the corresponding number of ListCells. Then the ListCell contents, that is, ListCell data 1 to n in FIG. 3, are respectively filled in the corresponding ListCells for rendering and displaying. As such, the memory and time needed for creating and arranging new ListCells can be reduced to improve the performance.

[0044] In an example implementation of the method reusing a ListCell in a hybrid

application in the present application, obtaining the ListCell by copying the ListCell template includes the following: traversing and obtaining all sub-views of the ListCell template; and creating a ListCell that includes all the sub-views and attributes of the sub-views. Here, all the sub-views (view) of the ListCell template are recursively
5 traversed, and then a ListCell is obtained through copying to include all the sub-views and the attributes corresponding to the sub-views. As such, the ListCell can be accurately copied. As shown in FIG. 4, the ListCell template, that is, sub-views of a ListCell prototype in FIG. 4, includes A, B, C, D, and E. Copying the ListCell prototype is a process of creating a ListCell that includes all the sub-views of the
10 ListCell prototype and attributes of the sub-views. Sub-views of the obtained ListCell include A', B', C', D', and E'.

[0045] In an example implementation of the method reusing a ListCell in a hybrid application in the present application, filling an obtained ListCell content in the copied ListCell includes the following: filling the obtained ListCell content in each
15 corresponding sub-view of the created ListCell. Here, the ListCell content can be filled in each corresponding sub-view of the copied ListCell based on the ListCell content. As such, the ListCell content can be accurately displayed at a corresponding location in the ListCell. As shown in FIG. 4, in the present implementation, the ListCell contents, that is, content data in FIG. 4, are respectively filled in the
20 corresponding sub-views A', B', C', D', and E' of the copied ListCell.

[0046] In an example implementation of the method reusing a ListCell in a hybrid application in the present application, the ListCell content is obtained from the front end. Here, as shown in FIG. 3, in addition to defining the prototype of the ListCell, the front end only needs to transmit the ListCell content, that is, the ListCell data in FIG.
25 3, to the native end. As such, memory and time consumed for front-end data processing can be reduced.

[0047] In an example implementation of the method reusing a ListCell in a hybrid application in the present application, during obtaining the ListCell by copying the ListCell template, the number of copied ListCells is determined based on the number
30 of ListCells that can be included and displayed in a display area. Here, especially when a ListCell is displayed in the display area for the first time, because there is no reusable ListCell yet, the corresponding number of ListCells need to be obtained through copying based on the number of ListCells that can be included and displayed in the display area. For example, in FIG. 3, the corresponding number of ListCells is

determined to be generated based on the number of ListCells that can be displayed on one screen. As such, the ListCell can be copied as needed, and the memory and time needed for creating and arranging new ListCells can be reduced to improve the performance. For example, 10 ListCell contents need to be displayed, and a display area can include and display 10 ListCells. In this case, 10 ListCells need to be copied.

[0048] In an example implementation of the method reusing a ListCell in a hybrid application in the present application, the reusing the ListCell includes the following: recycling a ListCell including the moved-out ListCell content as a reusable ListCell when a ListCell content is moved out of the display area; determining whether there is a reusable ListCell when a new ListCell content is to be moved to the display area; and if yes, filling the new ListCell content in the reusable ListCell, and displaying the ListCell; or if no, obtaining the ListCell by copying the ListCell template, filling the new ListCell content in the copied ListCell, and displaying the ListCell. Here, as shown in FIG. 3, especially when a ListCell is displayed in the display area for the first time, because there may already be a reusable ListCell at this time, the reusable ListCell can be used when a new ListCell content is displayed, that is, the ListCells in the display area are cyclically reused and moved in a list scrolling process. As such, the memory and time needed for creating and arranging new ListCells can be further reduced to improve the performance. Certainly, if there is no reusable ListCell, the ListCell template needs to be copied based on the new ListCell content to obtain a corresponding data ListCell. For example, 15 ListCell contents need to be displayed, and a display area can include and display 10 ListCells. In this case, 10 ListCells need to be copied in the first time of displaying, to display 10 ListCell contents. Five corresponding ListCell contents that have been displayed are moved out of the display area when the other five ListCell contents need to be moved to the display area for displaying. The five ListCells that are moved out of the display area can be recycled as reusable ListCells. Then, the five reusable ListCells can be reused for the other five ListCell contents.

[0049] As shown in FIG. 5, to better understand the principles and procedures of the present application, a hybrid framework (React Native) is used as an example to explain the implementation principles and procedures of the present application, which are as follows:

[0050] Step S51: A front-end layer defines a structure and a class of a ListCell.

[0051] Step S52: The front-end layer obtains list content data, that is, a ListCell

content.

[0052] Step S53: The front-end layer creates a ListCell prototype based on the defined structure and class of the ListCell, transmits the list content data to a native end, and then goes to step S56.

5 **[0053]** Step S56: A native-end layer creates a ListCell prototype instance, that is, a ListCell template, based on the ListCell prototype created by the front-end layer.

[0054] Step S57: The native end determines that a new ListCell is to be moved to a display area, and goes to step S58.

[0055] Step S62 is performed at the same time as step S57: The native end
10 determines that a ListCell is to be moved out of the display area, and goes to step S63.

[0056] Step S63: The native end uses the ListCell that is in the mobile display area as a reusable ListCell, and then goes to step S58.

[0057] Step S58: The native end determines whether there is a reusable ListCell.

[0058] If yes, step S59 is performed: The native end directly reuses the reusable
15 ListCell, and then goes to step S61. Here, the native end determines ListCells that can be reused based on the number of ListCells that can be included and displayed in the displayable area.

[0059] If no, step S60 is performed: The native end copies a ListCell from the ListCell prototype instance based on the list content data, and then goes to step S61.

20 Here, the native end copies other ListCells based on the list content data transmitted from the front-end layer.

[0060] Step S61: Render the ListCell, and then go to step S54.

[0061] Step S54: A front-end layer user scrolls a list.

[0062] Step S55: After triggering refreshing, the front-end layer goes to step S53.
25 In this case, the ListCell prototype does not need to be created again, and only refreshed content data of the list is transmitted to the native end, that is, new list content data to be displayed.

[0063] As shown in FIG. 6, according to another aspect of the present application, an apparatus reusing a ListCell in a hybrid application is further provided. The
30 apparatus 100 includes: a definition apparatus 1, configured to define a prototype of a ListCell at a front end of the hybrid application; and a copy and reuse apparatus 2, configured to obtain the ListCell through copying at a native end of the hybrid application based on the prototype of the ListCell, and reuse the ListCell. In the present application, the prototype of the ListCell is defined at the front end. The

prototype of the ListCell is copied at a native end (Native). The native end reuses an invisible ListCell based on a current list presentation location. As such, the prototype of the ListCell can be defined at the front end, and the ListCell can be freely created, rendered, and reused at the native end. Therefore, not only the flexibility of defining the ListCell in the hybrid development can be reserved, but also the ListCell can be reused at the native end, thereby greatly improving the large-list presentation and operation performance. The core of the technical solution of the present application is to resolve the problem that a structure and a class of a ListCell are defined at a front end but the ListCell is created and reused at a native end in the hybrid development. As such, the ListCell can be reused on a list page to improve the rendering and operation performance. The present application is applicable to all hybrid APP developments, and has a very good implementation effect in a React Native framework.

[0064] In an example implementation of the apparatus reusing a ListCell in a hybrid application in the present application, the definition apparatus 1 is configured to define a structure layout and a class of the prototype of the ListCell. Here, defining a prototype of a ListCell includes defining a structure and a class of the ListCell. As such, the ListCell can be flexibly defined at the front end in the hybrid development.

[0065] In an example implementation of the apparatus reusing a ListCell in a hybrid application in the present application, the prototype of the ListCell is a prototype of a single ListCell. Here, only one instance (that is, the prototype of the ListCell) is created at a front-end layer. Subsequently, a ListCell template can be created at the native end only based on the prototype. Then the ListCell template is copied as needed. As such, memory and time needed for creating and arranging new ListCells can be reduced to improve the performance.

[0066] In an example implementation of the apparatus reusing a ListCell in a hybrid application in the present application, the copy and reuse apparatus 2 is configured to create a ListCell template based on the prototype of the ListCell, obtain a ListCell by copying the ListCell template, fill an obtained ListCell content in the copied ListCell, and display the ListCell. Here, as shown in FIG. 3, the ListCell template, that is, a ListCell prototype instance in the figure, is created at the native end only based on the prototype defined at the front end. Then the ListCell template is copied based on the number of ListCell contents that need to be displayed, to obtain the corresponding number of ListCells. Then the ListCell contents, that is, ListCell

data 1 to n in FIG. 3, are respectively filled in the corresponding ListCells for rendering and displaying. As such, the memory and time needed for creating and arranging new ListCells can be reduced to improve the performance.

[0067] In an example implementation of the apparatus reusing a ListCell in a hybrid application in the present application, the copy and reuse apparatus 2 is configured to traverse and obtain all sub-views of the ListCell template, and create a ListCell that includes all the sub-views and attributes of the sub-views. Here, all the sub-views (view) of the ListCell template are recursively traversed, and then a ListCell is obtained through copying to include all the sub-views and the attributes corresponding to the sub-views. As such, the ListCell can be accurately copied. As shown in FIG. 4, the ListCell template, that is, sub-views of a ListCell prototype in FIG. 4, includes A, B, C, D, and E. Copying the ListCell prototype is a process of creating a ListCell that includes all the sub-views of the ListCell prototype and attributes of the sub-views. Sub-views of the obtained ListCell include A', B', C', D', and E'.

[0068] In an example implementation of the apparatus reusing a ListCell in a hybrid application in the present application, the copy and reuse apparatus 2 is configured to fill the obtained ListCell content in each corresponding sub-view of the created ListCell. Here, the ListCell content can be filled in each corresponding sub-view of the copied ListCell based on the ListCell content. As such, the ListCell content can be accurately displayed at a corresponding location in the ListCell. As shown in FIG. 4, in the present implementation, the ListCell contents, that is, content data in FIG. 4, are respectively filled in the corresponding sub-views A', B', C', D', and E' of the copied ListCell.

[0069] In an example implementation of the apparatus reusing a ListCell in a hybrid application in the present application, the ListCell content is obtained from the front end. Here, as shown in FIG. 3, in addition to defining the prototype of the ListCell, the front end only needs to transmit the ListCell content, that is, the ListCell data in FIG. 3, to the native end. As such, memory and time consumed for front-end data processing can be reduced.

[0070] In an example implementation of the apparatus reusing a ListCell in a hybrid application in the present application, the copy and reuse apparatus 2 is configured to determine the number of copied ListCells based on the number of ListCells that can be included and displayed in a display area. Here, especially when a

ListCell is displayed in the display area for the first time, because there is no reusable ListCell yet, the corresponding number of ListCells need to be obtained through copying based on the number of ListCells that can be included and displayed in the display area. For example, in FIG. 3, the corresponding number of ListCells is
5 determined to be generated based on the number of ListCells that can be displayed on one screen. As such, the ListCell can be copied as needed, and the memory and time needed for creating and arranging new ListCells can be reduced to improve the performance. For example, 10 ListCell contents need to be displayed, and a display area can include and display 10 ListCells. In this case, 10 ListCells need to be copied.

10 **[0071]** In an example implementation of the apparatus reusing a ListCell in a hybrid application in the present application, the copy and reuse apparatus 2 is configured to recycle a ListCell including the moved-out ListCell content as a reusable ListCell when a ListCell content is moved out of the display area; determine
15 whether there is a reusable ListCell when a new ListCell content is to be moved to the display area; and if yes, fill the new ListCell content in the reusable ListCell, and display the ListCell; or if no, obtain the ListCell by copying the ListCell template, fill the new ListCell content in the copied ListCell, and display the ListCell. Here, as shown in FIG. 3, especially when a ListCell is displayed in the display area for the first time, because there may already be a reusable ListCell at this time, the reusable
20 ListCell can be used when a new ListCell content is displayed, that is, the ListCells in the display area are cyclically reused and moved in a list scrolling process. As such, the memory and time needed for creating and arranging new ListCells can be further reduced to improve the performance. Certainly, if there is no reusable ListCell, the ListCell template needs to be copied based on the new ListCell content to obtain a
25 corresponding data ListCell. For example, 15 ListCell contents need to be displayed, and a display area can include and display 10 ListCells. In this case, 10 ListCells need to be copied in the first time of displaying, to display 10 ListCell contents. Five corresponding ListCell contents that have been displayed are moved out of the display area when the other five ListCell contents need to be moved to the display area for
30 displaying. The five ListCells that are moved out of the display area can be recycled as reusable ListCells. The five reusable ListCells can be reused for the other five

ListCell contents.

[0072] In conclusion, in the present application, the prototype of the ListCell is defined at the front end. The prototype of the ListCell is copied at the native end. The native end reuses an invisible ListCell based on a current list presentation location. As such, the prototype of the ListCell can be defined at the front end, and the ListCell can be freely created, rendered, and reused at the native end. Therefore, not only the flexibility of defining the ListCell in the hybrid development can be reserved, but also the ListCell can be reused at the native end, thereby greatly improving the large-list presentation and operation performance. The core of the technical solution of the present application is to resolve the problem that a structure and a class of a ListCell are defined at a front end but the ListCell is created and reused at a native end in the hybrid application development, so as to reuse the ListCell on a list page and improve rendering and operation performance.

[0073] Apparently, a person skilled in the art can make various modifications and variations to the present application without departing from the spirit and scope of the present application. The present application is intended to cover these modifications and variations of the present application provided that they fall within the scope of protection defined by the following claims and their equivalent technologies.

[0074] It is worthwhile to note that the present application can be implemented in software and/or a combination of software and hardware. For example, the present application can be implemented by using an application-specific integrated circuit (ASIC), a general purpose computer, or any other similar hardware device. In an implementation, a software program in the present application can be executed by a processor, so as to implement the previous steps and functions. Similarly, the software program in the present application (including a related data structure) can be stored in a computer readable recording medium, for example, a RAM memory, a magnetic or optical drive, or a floppy disk and a similar device. In addition, some steps or functions in the present application can be implemented by using hardware, for example, a circuit that cooperates with the processor to execute the steps or the functions.

[0075] In addition, a part of the present application can be implemented as a computer program product, such as a computer program instruction. When the computer program instruction is executed by a computer, the methods and/or the

technical solutions based on the present application can be invoked or provided by using operations of the computer. In addition, the program instruction for invoking the method in the present application can be stored in a fixed or removable recording medium, and/or is transmitted by using broadcast or a data flow in other signal bearer
5 media, and/or is stored in an operating memory of a computer device that operates based on the program instruction. Here, an implementation according to the present application includes an apparatus, and the apparatus includes a memory configured to store a computer program instruction and a processor configured to execute the program instruction. When the computer program instruction is executed by the
10 processor, the computer program instruction triggers the apparatus to operate based on the preceding methods and/or technical solutions in a plurality of implementations according to the present application.

[0076] It is apparent to a person skilled in the art that the present application is not limited to the details of the previous example implementations, and the present
15 application can be implemented in another specific form without departing from the spirit or essential feature of the present application. Therefore, the implementations should be considered as illustrative and not restrictive in every aspect, and the scope of the present application is limited by the appended claims, rather than the above description. Therefore, the present application covers all changes within the meaning
20 and the scope of the equivalent elements of the claims. Any reference numeral in the claims should not be considered as a limitation on the related claims. In addition, it is clear that the word "include" does not exclude another unit or step, and the singular does not exclude the plural. A plurality of units or apparatuses described in the apparatus claims can also be implemented by one unit or apparatus by using software
25 or hardware. The words such as "first" and "second" are used to indicate names instead of any particular order.

CLAIMS

1. A computer-implemented method, comprising:
 - defining, by a front end layer of a hybrid software application, a prototype of a ListCell;
 - creating, by a native layer of the hybrid software application, a ListCell template based on a created prototype of the ListCell defined by the front end layer;
 - obtaining, by the native layer, a ListCell by copying the ListCell template;
 - obtaining, by the native layer, first ListCell content transmitted from the front end layer;
 - filling, by the native layer, the ListCell with the obtained first ListCell content;
 - initiating, by the native layer, display of the ListCell having the first ListCell content; and
 - reusing, by the native layer, the ListCell, comprising:
 - in response to a user input, determining that the first ListCell content has been moved out of a display area of the native layer,
 - in response, removing the first ListCell content from the ListCell, obtaining second ListCell content transmitted from the front end layer, and
 - refilling the ListCell with the second ListCell content.
2. The computer-implemented method according to claim 1, wherein defining the prototype of the ListCell comprises defining a structure layout and a class of the prototype of the ListCell.
3. The computer-implemented method according to claim 1, wherein obtaining the ListCell comprises:
 - traversing and obtaining all sub-views of the ListCell template; and
 - creating a ListCell that comprises all the sub-views and attributes of the sub-views.
4. The computer-implemented method according to claim 3, wherein filling the ListCell comprises filling the obtained first ListCell content in each corresponding sub-view of the created ListCell, wherein the obtained first ListCell content is obtained from the front end of the hybrid software application.

5. The computer-implemented method according to claim 1, comprising, during obtaining the ListCell by copying the ListCell template, a number of copied ListCells is determined based on a number of ListCells that can be displayed in a computer display area.
6. A non-transitory, computer-readable medium storing one or more instructions executable by a computer system to perform operations according to the method of any one of claims 1 to 5.
7. A computer-implemented system, comprising:
 - one or more computers; and
 - one or more computer memory devices interoperably coupled with the one or more computers and having tangible, non-transitory, machine-readable media storing one or more instructions that, when executed by the one or more computers, perform one or more operations according to the method of any one of claims 1 to 5.