



智慧家庭 App 的设备控制页面动态呈现技术

郭涛, 韩建亭, 杨晓玲

(中国电信股份有限公司上海研究院, 上海 200122)

摘要: 对智慧家庭 App 进行设备控制的页面动态呈现技术进行了分析, 提出了一种基于轻量级部件引擎的页面动态呈现技术, 并结合实际案例进行了开发验证, 对优化智慧家庭 App 设备控制的页面动态呈现技术进行了有益的探索。

关键词: 智能部件引擎; 轻量级; 智慧家庭; App; 动态呈现

中图分类号: TP311.5

文献标识码: A

doi: 10.11959/j.issn.1000-0801.2017293

Key technique of dynamic interface rendering based on device control of smart home App

GUO Tao, HAN Jianting, YANG Xiaoling

Shanghai Research Institute of China Telecom Co., Ltd., Shanghai 200122, China

Abstract: The dynamic interface rendering technology of smart home App was analyzed, and a dynamic interface rendering method based on SWE (smart widget engine) was proposed. Based on an actual case, a smart home App was developed to verify SWE. It shows the method of SWE is a useful exploration for optimization of dynamic interface rendering technology.

Key words: SWE, lightweight, smart home, App, dynamic interface rendering

1 引言

在家庭终端智能化和万物互联的发展趋势下, 智慧家庭业务成为行业热点。在智慧家庭系统中, 智慧家庭 App 是一个重要的控制终端。除了对传统的家庭网关、机顶盒进行业务管理外, 智慧家庭 App 要重新考虑如何面对家庭内形态各异的智能终端进行管理、控制, 这对 App 的设备管理技术有了更高的要求。一方面智慧家庭 App 在支持功能属性各异、品类繁杂的各种终端

设备的同时, 既要求非常灵活及时地调整设备界面信息和状态, 也要确保客户端的体积不能过大。如果采用 App 原生技术进行编码, 每次添加一个设备都需要增加相应的代码, 同时还要通知用户更新客户端, 这不仅非常的繁琐, 而且还导致客户端大小无限制增长, 最终可能导致用户下载安装繁琐。另一方面为了更好的用户体验, 特别当用户登录 App 智能设备控制首页面, 需要在一个页面加载呈现多个设备的控制卡片页时, 页面呈现的速度是影响用户感知的重要因素。因此面向



设备控制的智慧家庭 App 的动态呈现技术成为关键的技术研究点。目前业界在开发针对智能设备控制的手机 App 时，设备控制页面加载与渲染大多基于 HTML5（下文简称 H5）的浏览器架构进行开发，业界也有 React Native（下文简称 RN）、WEEX 等移动端开源平台开发框架，致力于确保 App 本身的灵活性、迭代更新的周期和成本优化到极致。本文对智慧家庭 App 进行设备控制的页面动态呈现技术进行了对比和优劣势分析，尝试提出一种基于轻量级部件引擎的页面动态呈现技术作为现有技术方案的补充，并结合实际案例进行了开发验证，对优化智慧家庭 App 设备控制的页面呈现技术进行了有益的探索。

2 设备控制页面动态呈现技术

针对设备控制页面动态呈现技术，目前可参考的移动开源框架和技术主要有 3 种，分别是 Hybrid、RN 和 WEEX，这 3 种方案针对解决客户端版本频繁更新和多端研发两大痛点，同时力求解决前端语言性能差和显示效果受限的问题。

（1）Hybrid 方案

Hybrid 是混合模式应用的简称，拥有跨平台特性，可以进行内容“热更新”，不受限于审核、上线等繁复的流程，是应用较多的移动开放框架，如图 1 所示。其原理就是以 Webview 为容器，以 HTML5 为基石，通过定义 Native 特性的扩展来支持动态化产品研发^[1]，即在 App 中嵌入一个 Webview 组件，可以在这个组件中载入页面，相当于内嵌的浏览器^[2]。Hybrid App 由 Webview 作为桥梁，负责 Native 调用前端页面的 JavaScript（下文简称 JS）方法，或者前端页面通过 JS 调用 Native 提供的接口。Hybrid 方案解决了跨平台、开发效率高、节约开发成本等问题，但因为 Webview 的实现是网页开发思路，因此界面渲染效率低，多线程支持差，动画展现效果、页面渲

染速度等性能方面比 Native 应用还有很大差距，不太适合应用在要求动态效果和性能体验较高的 App 首页面或频道主页面。

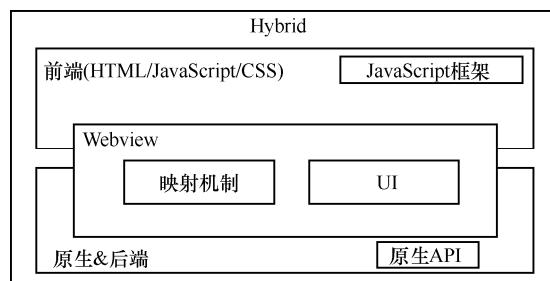


图 1 Hybrid 方案原理

（2）RN 方案

RN 是 Facebook 基于 React 开发的跨平台用户交互界面（UI）框架，以 Native 为渲染引擎，通过脚本引擎支持界面虚拟文档对象模型（virtual DOM）的转换和逻辑控制，实现界面的动态加载和呈现^[3]，如图 2 所示。也就是说 RN 是利用 JS 内核转化成 Native 运行，体验与原生开发框架更接近，性能相对也更好。RN 继承了 React 在 JSX（JavaScript 的扩展语法）中直接以声明式的方式来描述 UI 结构的机制，提高了 UI 渲染效率。对于一个用 RN 搭建的移动 App，在启动后会从服务器下载最新的 JS 绑定文件，然后由本地 JS 内核引擎对 JS 文件进行解析，并利用映射机制转换为对应的 Native 方法和 UI 控件。在实际开发中，要做到最佳用户体验，针对 iOS 和 Android 还是要分别编写 UI 代码。目前 RN 的技术方案也在不断迭代和演进过程中，与 Hybrid 方案相比，RN 在 Native 端提供更强大的控件调用能力，因此更具扩展性、灵活性和性能优势。

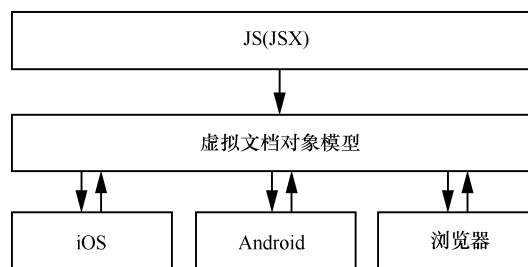


图 2 RN 方案原理

(3) WEEX 方案

WEEX 是阿里公司提出的一种轻量级的移动端跨平台动态性技术解决方案,如图 3 所示。WEEX 方案致力于充分调动 Native 能力,充分解决和回避性能瓶颈,保持较低的开发成本和学习成本,本质上 WEEX 与 RN 实现原理类似。与 RN 采用 JavaScript XML(JSX)语法开发不同,WEEX 是基于 Vue(JS 框架)的语法开发页面或组件,然后将它们转换为 JS 绑定以进行部署。当服务器返回给客户端 JS 绑定时,JS 绑定会被客户端的 JavaScript 引擎处理,并管理渲染 Native 视图,调用原生 API 和用户交互,如图 3 所示。

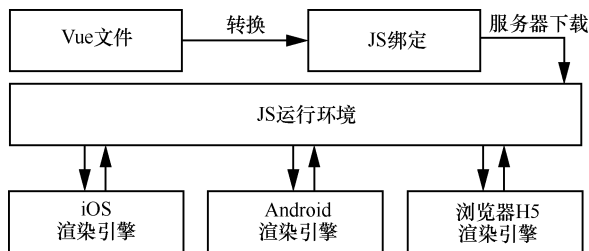


图 3 WEEX 方案原理

综上所述,Hybrid、RN 和 WEEX 都可作为动态内容呈现的技术方案。与 Hybrid 的网页开发思路不同,RN 和 WEEX 为了获得跨平台原生 App 一致性体验,更加强调对 Native 能力的调用,因此拥有更好的性能和流畅性体验,也成为目前移动 App 开发的热点技术。但是 RN 和 WEEX 目前也主要在 App 的部分页面进行了应用,如淘宝、天猫等在营销活动和频道首界面进行应用,仍无法代替原生 App 开发。基于本文提到的针对智能设备控制页面动态呈现的技术实现,RN 和 WEEX 虽然可以提供可参考的开发框架,但一方面 RN 和 WEEX 作为开源框架,长期的演进和支持力度存在不确定性,后续运营存在风险,一方面对运营商开发人员来说,如果在现有 App 中仅考虑设备控制页面的动态呈现问题,RN 和 WEEX 仍然有很大的前后端开发技术学习成本,量级偏重。

3 基于 SWE 的轻量级页面动态呈现技术

3.1 设计思路

基于智能部件引擎 SWE (smart widget engine) 的轻量级页面动态呈现技术是通过 XML 语言自定义视图引擎和逻辑引擎,并将单个设备数据根据设备 ID 和设备类型独立组织成轻量级设备容器,实现多页面独立逻辑、同步渲染,解决由一个内核处理渲染多个设备页面,并实现快速、流畅的交互体验问题。

以卡片式设备预览与快速控制页面(如图 4 所示)为例,需要在一个页面同时加载呈现多个设备页面,且各设备页面拥有独立的业务逻辑和设备属性。如果采用多个 H5 内核处理显示多个卡片(一一对应),则渲染效率太低,容易出现页面不流畅、加载失败等问题;如果采用一个 H5 内核处理多个设备页面,由于设备变量重名问题,容易出现逻辑混乱。因此需要采用一种新的多卡片动态渲染技术,保证多卡片同时快速渲染,并灵活、动态支持现有以及未来加入的各种终端智能设备的页面呈现,已提高客户端的用户体验。



图 4 卡片式设备预览与快速控制页面

3.2 技术架构

基于 SWE 的轻量级页面动态呈现技术定义



了一套描述视图结构的 XML (extensible markup language, 可扩展标记语言) 数据格式, 然后通过逻辑引擎、数据引擎、视图引擎和支撑模块来达到 Native 级别的动态化界面效果, 架构如图 4 所示。当 App 启动后, 首先从服务器下载 SWE 资源压缩包, 资源包中包含设备页面动态呈现所需要的 XML 描述文件以及相应的图片资源、国际化资源等信息。下载完成后, 由支撑模块对资源包解压缩并读取其中的 XML 描述文件, 并分别初始化视图引擎和逻辑引擎。视图引擎主要负责读取 XML 中定义的视图内容和视图呈现方式, 然后通过本地 Native 方式进行渲染呈现出来, 逻辑引擎主要是读取 XML 中视图的交互和控件的动作逻辑, 以便可以及时响应用户对视图的操作, 而数据引擎则负责与服务器进行通信, 最终由数据来驱动视图和逻辑的变化, 三者各司其职, 以数据为支撑, 对业务逻辑进行分析并对视图进行正确的渲染。

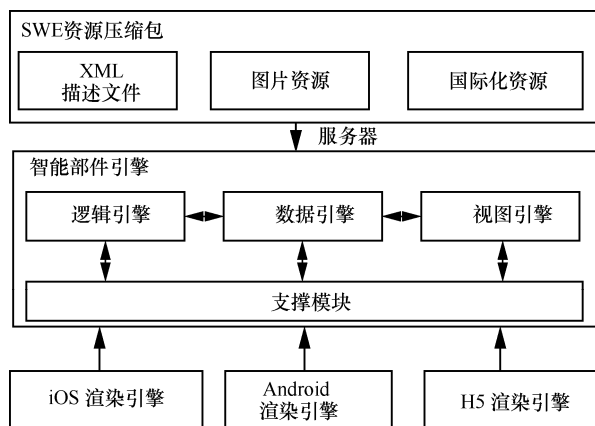


图 4 基于 SWE 轻量级页面动态呈现技术架构

基于 SWE 的轻量级页面动态呈现技术包括逻辑引擎、视图引擎、数据引擎与支撑模块 4 个模块, 各模块的功能如下。

- 视图引擎用于渲染设备页面, 主要负责视图呈现。视图引擎的设计原理是, 基于部件进行组合渲染, 其内部封装了系统原生控件, 采用绝对坐标定位方式,

结合当前屏幕比例智能进行适配的策略, 因此速度渲染速度极快, 有着良好的用户体验。

- 逻辑引擎主要用于监听用户的动作行为 (比如点击事件、滑动事件、长按事件等) 以及对事件逻辑进行一定的逻辑和算术运算等, 包括动作逻辑控件、事件监听模块、逻辑运算模块与算术运算模块。根据用户的不同动作请求, 结合配置文件中的业务逻辑执行相应的动作。用户的动作行为有可能导致视图发生变化, 也可能会引发底层向对应的设备发送指令, 当指令返回结果时, 间接响应视图发生变化。
- 数据引擎主要是用于处理设备相关的状态数据。由于 SWE 的方案设计的核心是基于数据驱动模型 (视图的变化是随着数据的变化而发生变化, 并不是随着用户的动作而发生变化), 因此数据模块尤其重要。数据模块采用分层的设计原型, 从下而上依次是 TCP 通信层、协议封装层、数据缓冲层、数据驱动层。其中数据驱动层主要负责支撑视图引擎和逻辑引擎。
- 支撑模块主要是为视图引擎、逻辑引擎、数据引擎提供底层支撑, 包括 I18N 的国际化支持、XML 解析和加载相关的支撑、图像加载的支撑、屏幕自动适配的支撑以及资源包管理的支撑。

3.3 使用方法

由于 SWE 的轻量级部件渲染技术是基于 XML 技术, 因此开发者只需要了解 XML 的配置语法和内置的配置标签即可根据需求完成客户端设备页面的开发。

3.3.1 资源包构成

每一个应用对应一个资源包, 资源包中包含两个 XML 文件、一组图像文件、一组国际化文件等。最终完成之后的目录结构如图 5 所示。

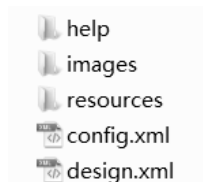


图 5 目录结构

- **design.xml**: 设备的界面和功能配置文件。
- **config.xml**: 设备的卡片配置文件。
- **resources** 目录: 存放所有支持的国际化文件。比如需要支持中文和英文, 则需要在该目录生成两个文件, 分别命名为 **zh_CN.properties** 和 **en_US.properties**, zh 和 en 是国际上通用的标准语言代码缩写, CN 和 US 是国际上通用的标准国家代码缩写。
- **help** 目录: 存放设备帮助信息界面所用到的资源和文件。该部分内容以网页方式提供, 提供一个 **index.html** 的主页面以及相应的子页面链接和用的资源, 如图 6 所示。

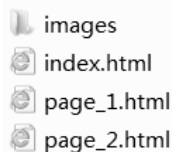


图 6 资源和文件

- **images** 目录: 存放界面中所有用到的图片资源文件。

3.3.2 标签介绍

完成配置的开发需要涉及较多 XML 标签, 标签定义如下。

- **基础控件标签**: 定义常用的基本控件类型, 如 **view**、**button**、**text**、**edit**、**image**、**number**、**progress** 等, 这些标签作为界面上的功能展现和响应者, 一般不能包含其他子控件, 仅可以包含监听器控件标签。
- **布局控件标签**: 定义界面布局的相关标签, 包括 **root**、**linear**、**frame**、**block**、**dialog**、**commonDialog**、**drop**、**page** 等。此类型标

签中可以嵌套布局控件标签、基础控件标签以及监听器控件标签。

- **监听器标签**: 用于捕获控件型及布局型标签的各种用户操作事件, 如点击事件、滑动事件等。标签显示为 **listener**, **listener** 标签为定义的监听事件, 嵌套在其他标签下并对其事件进行监听。
- **动作标签**: 用于执行具体的动作行为, 比如监听到用户的点击事情后, 可以调用底层通信层发送执行, 也可以更改页面逻辑, 或者进行页面跳转等。
- **关系型标签**: 定义页面的值关系脚本, 该标签的定义为 **relation**, 其中定义了一组属性值, 用于定义页面的值和控件的关系, 比如当编号为 1 的属性 (开关机属性), 其值为 1 时表示电源开, 为 0 时表示电源关闭, 此时需要对该属性进行配置, 使属性值和界面关联起来, 最终达到数据驱动界面的效果。

4 方案验证

4.1 方案实施

基于本技术方案, 封装智能设备控制 SDK 并进行了 App 集成验证。智慧家庭 App 中新增智能家居设备状态呈现和控制首界面 Tab 页, 实现对多个智能设备状态和控制页面的统一呈现, 点击单个设备后, 进入设备的详细状态和控制页面。针对智能家居 Tab 页的具体业务流程如图 7 所示。

步骤 1 用户进入客户端的“智能家居”Tab 页, 初始化智能家居 SDK。

步骤 2 调用 SDK 登录接口, 登录成功后 SDK 会自动下载用户的设备列表, 并根据返回的设备信息拼接设备 zip 压缩包包名, 根据该包名下载对应的 XML 配置压缩包。

- 如果设备的压缩包不存在, 则下载资源, 如果设备的压缩包存在, 则检查是否有更

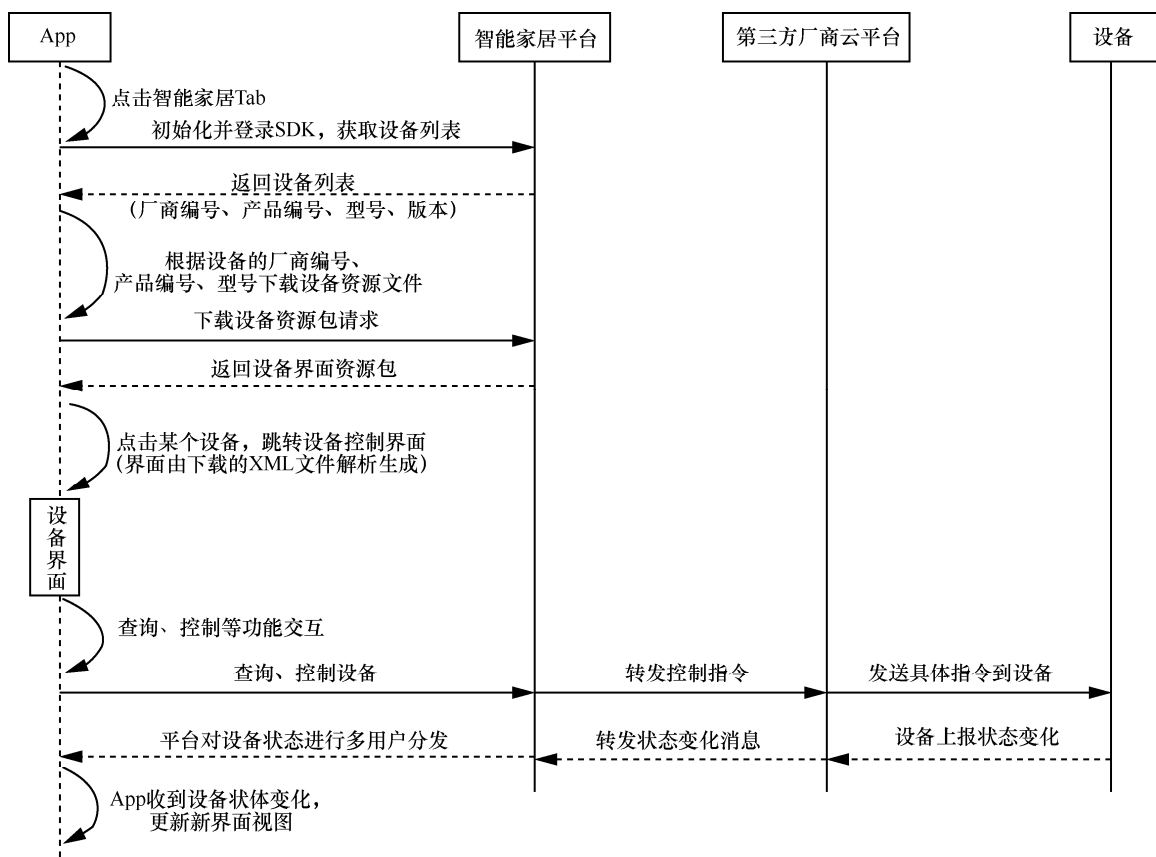


图 7 针对智能家居 Tab 页的具体业务流程

新，如果有更新又分为版本号更新和 MD5 更新两种情况：

- 如果版本号不一致，则直接下载最新资源，重新解压；
- 如果版本号一致，则匹配 MD5 值，判断是否当前版本有改动，如果有改动也重新下载资源解压。
- 解压之后检查设备配置目录下的配置文件是否缺失，如果缺失则尝试解压旧的文件。

步骤 3 点击设备，通过下载的资源文件解析生成设备详情界面。设备查询控制相关功能通过智能家居云平台和第三方厂商交互返回，App 收到设备状态后更新界面。

4.2 具体页面配置实例

以空调的快速预览卡片为例，在整个界面中，控件的位置、图片等的大小以像素为单位。控件

的绝对坐标以所在布局的左上角为起点，到该控件中心点的距离，界面效果如图 8 所示。

卡片是在 config.xml 文件中配置，config 整体架构如下。

```
<?xml version="1.0" encoding="UTF-8"?>
<config logo="logo.png" guide="guide.png"
keepGuide="false">
  <root h="354" bkImg="card_bg" design-
Width="720" designHeight="1234">
    <block name="text_box" h="264">
    </block>
    <block h="88">
    </block>
    <relation>
    <dvid id="1002"/>
    </relation>
```

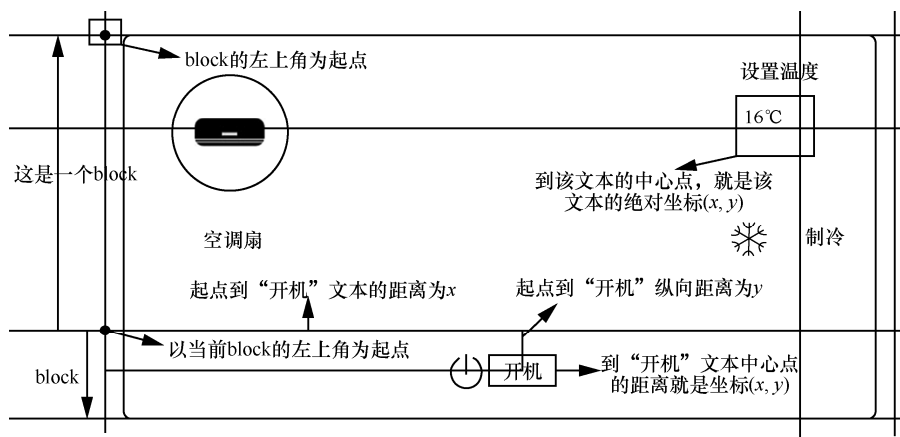


图8 界面效果

</root>

</config>

对比现在业界使用的 HTML5 页面渲染技术, 以空调的页面为例, 综合性能指标对比结果见表 1。

表 1 综合性能指标对比结果

指标	HTML5	SWE
页面加载速度/ms	101	31
页面渲染速度/ms	115	42
页面跳转速度/ms	203	68
动作响应时间/ms	5	2
页面开发周期/天	0.5	0.5

4.3 方案技术优势

本方案经验证, 具有以下技术优势。

- 实现了跨平台开发一致性体验, 升级维护简单, 大大减少了开发时间和开发成本。
- 结构化的界面描述, 支持同一个界面中对多个设备页面的动态加载和渲染呈现。
- 学习成本低, 只要了解简单的文本编辑, 就可以很快速地写出功能强大的设备界面和业务逻辑。
- 自适应能力强, 由于 Android 和 iOS 手机屏幕大小不一, 各种尺寸各种比例都有, 采用 SWE 框架可以很好适配各种分辨率的客户端。
- 极好的用户体验, 由于采用原生系统的解

析, 没有经过第三方的内核, 因此速度更快, App 操作更顺滑。

5 结束语

本文提出的基于轻量级部件引擎的页面动态呈现技术, 在针对智能设备控制页面的动态呈现技术方面做出了有益的探索, 经开发验证具有开发难度小, 支持跨平台部署和渲染速度快等优点, 可应用于实现多页面动态呈现维度的技术需求。此方案可作为 H5 技术的补充, 在实际开发中采用“SWE+H5”的组合开发方式, 利用 SWE 技术交互体验好的特点实现设备快速预览卡片の開発, 而利用 H5 简单易学的特点负责详细设备控制页面的开发, 保证客户端达到最好的设备控制动态呈现效果。

参考文献:

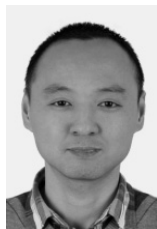
- [1] 郭涛, 韩建亭, 杨晓玲. 智能家居管理系统分析与技术架构[J]. 电信科学, 2015, 31(11): 106-111.
GUO T, HAN J T, YANG X L. Architecture and key capabilities of management system of smart home device[J]. Telecommunications Science, 2015, 31(11): 106-111.
- [2] 童丽霞, 何加铭, 陈恳, 等. 基于 HTML5 技术的 Widget 引擎内容缓存模型及实现[J]. 计算机应用研究, 2011, 28(12): 4625-4628.
TONG L X, HE J M, CHEN K, et al. HTML5-based content cache algorithm design and implementation on mobile Widget engine [J]. Application Research of Computers, 2011, 28(12): 4625-4628.



- [3] 刘文. 一种基于 XML 的工作流引擎设计和实现[J]. 电脑编程技巧与维护, 2016(16):18-19.

LIU W. Design and implementation of a workflow engine based on XML[J]. Computer Programming Skills & Maintenance, 2016(16):18-19.

[作者简介]



郭涛（1978-），男，中国电信股份有限公司上海研究院高级工程师，主要研究方向为家庭网络、智能终端和智能家居平台等。



韩建亭，男，中国电信股份有限公司上海研究院教授级高级工程师，主要研究方向为家庭网络、宽带接入、固网移动融合和移动互联网融合等。



杨晓玲，女，中国电信股份有限公司上海研究院工程师，主要研究方向为智能家居平台、家庭网络等。