一种基于 OSGi 的 Web 应用模块化架构设计①

刘 丹, 王宇飞, 杨 宁

(国网电力科学研究院 北京中电普华信息技术有限公司, 北京 100192)

摘 要: 针对 Web 应用在模块化和动态性方面的不足, 结合当前 Web 应用的关键技术点 Servlet、JSP、静态资源、请求处理等, 基于 OSGi 技术给出了 Web 应用的模块化架构, 包括 Web 应用与 OSGi 结合的技术路线, 引入 OSGi 之后应用结构和加载方式的改变, Web 应用构件 Servlet、Filter 和 Listener 的模块化拆分与注册, 模块中 JSP 的注册、获取、编译, 模块中资源文件的注册与获取, 请求的分发与处理. 使 Web 应用既能沿用已有的技术和框架, 又具备模块化和动态性.

关键词: OSGi; Web 应用; 模块化

Design of Modularized Architecture for Web Application Based on OSGi

LIU Dan, WANG Yu-Fei, YANG Ning

(Beijing China Power Information Technology Co. Ltd, State Grid Electric Power Research Institute, Beijing 100192, China)

Abstract: To deal with the deficiency of modularization and dynamics, this paper proposes an architecture for web application based on OSGi, combined with the popular technologies of web application, such as servlets, JSPs, static documents and request handling, including the technique of integrating web application with OSGi, the change of structure and loading of application. Then it illustrates how to modularize and register web application components including servlets, filters and listeners, how to register, find and compile JSPs, how to register and find static documents. Finally, it discusses the process of dispatching and handling requests. With the architecture, besides continuing to use the existing technologies and frameworks, web application can also have the characteristics of modularization and dynamics.

Key words: OSGi; Web application; modularize

Web 应用发展至今, 积累了丰富的技术和框架, 但是模块化和动态性难题却一直没有得到很好的解决. 虽然在 Web 应用的设计阶段可以采用模块化的思想来划分系统功能, 但是实现过程中却没有对应的技术手段使模块化得到落实和保障; 另外, Web 应用的动态性也非常有限, 除了动态修改配置项, 如果要对某个功能的实现代码进行更新, 通常需要重新启动整个应用或容器, 进而影响到在线用户. 另一方面, 设计初衷是为了嵌入式应用的 OSGi 规范, 却因为具备良好的模块化和动态性, 在 Java 领域引起了广泛关注, 并且已经在著名的集成开发工具 Eclipse 中得到了成功应用. 为了更好的支持企业级应用, OSGi 联盟特别成立了企业专家组, 并发布了 OSGi 规范的企业版本, 规定了如何将 Java EE 的关键技术集成到 OSGi 中, 其中与 Web 应用相关的有 HTTP 服务规范[1]和 Web 应用规范[1]. HTTP 服务规范实现了对 Servlet 规范和 HTTP 规范的支持, 但是缺少对 Filter、Listener 的支持, 这对于 Web 应用来说是不足的. 虽然有开源项目实现了 HTTP 服务规范, 但是仍然不完整. Web 应用规范定义了一个 Web 应用的打包、部署模型. 该规范主要是对应用服务器的实现提出了新的要求, 对于一般的应用开发团队或企业来说, 实现难度较大. 虽然已有开源项目和商业产品实现了该规范, 但是将涉及到应用服务器的升级或替换, 会增加企业应用开发成本. 本文将针对这些现状, 在不改变传统应用服务器的前提下, 补充完善 HTTP 服务规范的缺失, 以一种简单经济的

① 收稿时间:2013-06-19;收到修改稿时间:2013-07-11

方式实现 Web 应用与 OSGi 的结合. 使得 Web 应用即可以直接部署到传统应用服务器中, 又具备 OSGi 的模块性和动态性, 可以“即插即用, 即删即无”及运行时动态更新.

1 相关技术

1.1 Web 应用

Web 应用是 Web 服务器的动态扩展, Web 应用由 Web 构件(包括 Servlet 和 JSP)、Bean 构件、静态资源文件(例如: HTML、JS、CSS、图片、Flash 等)以及客户端 Applets 等元素构成[2]. Web 服务器会为运行在其之上的 Web 应用提供一个 Web 容器, Web 容器是 Web 构件的运行平台, 负责提供请求转发、安全、并发访问控制和生命周期管理等服务, 并向 Web 构件提供命名、事务的 API 接口. Web 应用有一套固定的目录结构, 不同类型的文件有严格的存放规则, 如表 1 所示.

表 1 Web 应用目录结构说明

目录名称	目录(文件)说明
WEB-INF/class	Java 类文件目录
WEB-INF/lib	应用的 Java 类包库目录
WEB-INF/web.xml	Web 应用部署描述符文件
(*.html, *.jsp, *.js, *.css, *.jpg, *.swf, …)	存放 html、jsp、js、css、图片等能直接通过浏览器访问的 Web 资源文件

1.2 OSGi

OSGi(Open Service Gateway Initiative, 开放服务网关协议)的初始目标是让服务提供商通过住宅网关, 为各种家庭智能设备提供服务, 例如: 通过 Web 页面控制咖啡机等. 后来逐渐成为一个为家庭自动化、交通工具、移动设备和其他环境下的网络设备的应用程序和服务进行传递和远程管理的开放式服务平台. OSGi 服务平台的核心规范[3]定义了一个优雅、完整和动态的模块(Bundle)模型, 模块的生命周期和服务注册管理. 模块无需重新引导就可以被远程安装、启动、更新和卸载(其中 Java 包 / 类的管理被详细定义). 服务注册允许模块去检测新服务和已注销的服务, 并执行相应的操作. OSGi 也被称为 Java 语言的动态模块系统, 为模块化应用的开发定义了一个基础架构. OSGi 规范的开源实现包括 Equinox、Felix、Knopflerfish 等.

1.3 OSGi 与 Web 应用

为了可以通过 Web 方式访问利用 HTTP、HTML、XML 和 Servlets 等技术实现的具有用户界面和交互操作的 OSGi 应用, OSGi 规范中包含了 HTTP 服务规范[1], 该服务实现了对 Servlet 规范和 HTTP 规范的支持. 开源项目 Equinox 实现了 HTTP 服务规范, 提供了一种基于 Eclipse 扩展点机制的动态注册、卸载 Servlet 和 Resource 映射的方法, 可以接收并处理 HTTP 请求.

为了更好的支持企业级应用 OSGi 联盟在 4.2 版本的企业规范中新增了 Web 应用规范[1], 该规范提供了一种将 Java EE 应用无缝部署到处于 OSGi 环境下的 Servlet 容器中的方法, 定义了 Web Application Bundle(WAB)作为 OSGi 环境下 Web 应用的打包、部署模型. 开源项目 Eclipse Virgo 以及商业产品 IBM Websphere、Sun Glassfish、JBoss AS 等实现了对 Web 应用规范的支持.

2 Web应用模块化架构

2.1 技术路线

Web 应用与 OSGi 的结合有两种方式[4]: 1)把 Web 应用嵌入到 OSGi 框架中, 如图 1 所示; 2)将 OSGi 框架嵌入到 Web 应用中, 如图 2 所示.

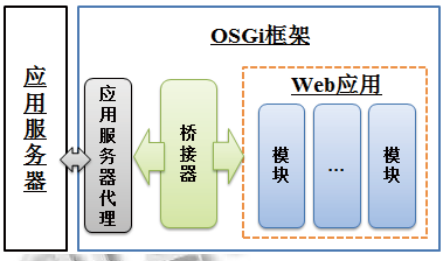

图 1 Web 应用嵌入 OSGi 框架

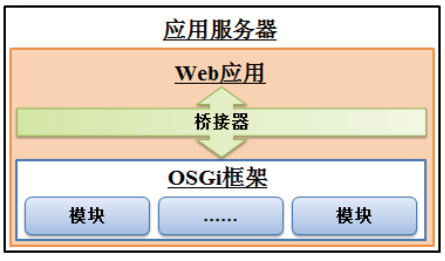

图 2 OSGi 框架嵌入 Web 应用

通过对比这两种方式可以发现, 第一种方式是一种完全的 OSGi 方式, 要求应用服务器或代理要能够以模块的方式运行在 OSGi 框架中, 或者把 Web 应用作为 OSGi 模块部署. 这种方式的优点是应用全部由模块组成, 可以充分利用 OSGi 的特性, 缺点是对应用

服务器和 Web 应用的改变比较大，对于普通的应用开发团队或企业来说实现难度较高，各应用服务器对 OSGi Web 应用规范的支持都是采用这种方式实现的. 第二种方式中应用从整体上还是一个标准的 Web 应用，可以传统方式部署并运行在应用服务器中，只是在应用内部实现模块化. 虽然 Web 应用自身无法模块化，会导致应用作为整体无法利用 OSGi 的动态更新特性，但是因为应用提供的业务功能都已经模块化，而通常变化更新最频繁的正是业务功能而不是应用整体框架，所以这种方式还是可以充分发挥 OSGi 特性的，而且这种方式实现难度也比较低，对应用服务器没有任何特殊要求. 因此本文选择采用第二种方式来实现 Web 应用的模块化架构.

2.2 应用结构

采用 OSGi 框架嵌入 Web 应用方式实现应用的模块化，应用从整体上还是一个标准的 Web 应用，具有固定的目录结构. 应用中没有具体内容，不再需要存放 Java 类文件和 Web 资源文件的相关目录；要在应用中集成 OSGi 框架，需要为模块化增加相关目录. 图 3 给出了模块化的 Web 应用的目录结构以及模块的目录结构，具体说明见表 2 和表 3.

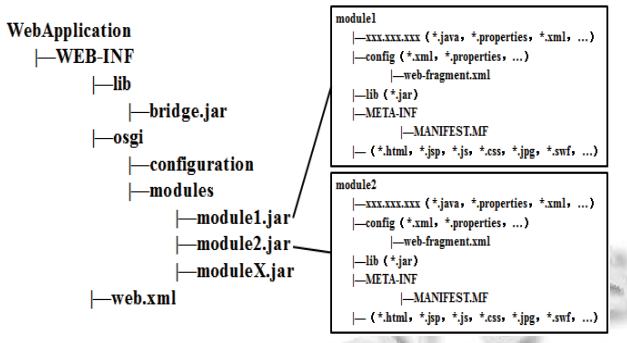

图 3　模块化的 Web 应用目录结构

表 2　Web 应用目录结构说明

目录名称	目录(文件)说明
WEB-INF/lib/bridge.jar	Web 应用与 OSGi 框架之间的桥接器，提供 Web 应用所需的 Java 类，同时负责启动 OSGi 框架
WEB-INF/osgi/configuration	OSGi 框架配置文件
WEB-INF/osgi/modules	模块集合
WEB-INF/web.xml	Web 应用部署描述符文件

对 Web 应用结构进行这样的调整后，与某个业务功能高度聚合的相关文件被划分到一个模块中，所有模块最终都以 JAR 包的形式集成在应用的特定目录中，模块不再只是逻辑上的独立单元，而是物理上的独立存在，更有利于复用.

表 3　模块目录结构说明

目录名称	目录(文件)说明
xxx.xxx.xxx	Java 类文件目录
config/web-fragment.xml	web.xml 片段(模拟标准 web.xml 中 Servlet、Filter、Listener 的配置)
lib	存放模块需要加载的以 JAR 包形式提供的类库
META-INF/ MANIFEST.MF	OSGi 模块标准配置文件
(*.html, *.jsp, *.js, *.css, *.jpg, *.swf, …)	存放页面相关文件，包括 html、jsp、js、css、图片等

2.3 加载方式

要实现 Web 应用与 OSGi 的结合，除了需要调整应用的目录结构外，OSGi 与 Web 应用在加载和管理各类资源的机制方面也存在很大的差异，如图 4 所示. 在传统 Web 应用中，应用的类加载器委托呈直线型关系，最底层的应用类加载器可以请求和访问所有的资源. 在 OSGi 应用中资源分布在不同模块，每个模块有独立的类加载器[3]，用来加载本模块中的资源，一个模块的类加载器可以委托另外一个模块的类加载器进行加载，所有模块的类加载器形成了一个类加载委托网络状结构.

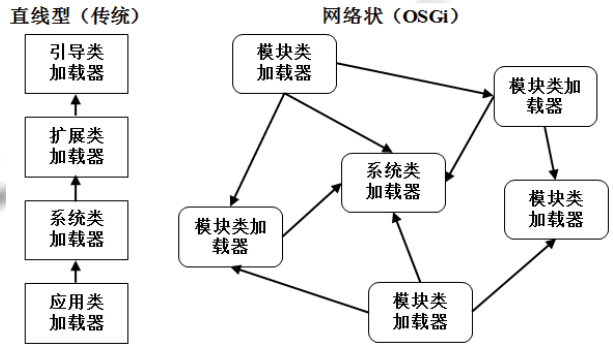

图 4　OSGi 与传统应用类加载委托模型对比

Java 类加载器的特性是类一旦被加载到内存中运行就无法被修改，正是这个特性导致我们无法在应用运行时修改 Java 类，而 OSGi 的类加载机制却巧妙的解决了这个问题. 当模块中的类有变更时，更新模块将释放所有已加载的类，并回收模块类加载器，然后再重新初始化一个类加载器，并重新加载模块中的类，从而实现了运行时以模块为单位的动态更新. 本设计正是利用了 OSGi 的这种类加载机制，实现了在 Web 应用运行

时对模块的更新, 使得 Web 应用具备了动态性.

OSGi 与传统应用类加载机制的不同, 除了给 Web 应用带来动态性的好处, 也带来一个问题, 就是资源的获取问题. 本设计将在 URL 请求的应用上下文路径后增加模块上下文路径, 用来定位到哪个模块中获取资源, 模块上下文路径之后是模块内的相对路径. 例如 http://localhost:8080/webapp/workbench/login.jsp, 应用上下文路径 webapp 后的/workbench 将被当作模块上下文路径来处理, 表示请求模块上下文路径为 workbench 的模块中的 login.jsp.

2.4 应用启动

随着 Web 应用结构的改变, 应用的启动过程也与传统方式有所不同. 传统方式下, 把 Web 应用部署到服务器之前, 需要在 Web 应用部署描述符文件即 web.xml 中对应用进行配置. 模块化的 Web 应用仍将保留 web.xml, 不同的是 web.xml 中配置的是由桥接器(bridge.jar)提供的几个固定的全局 Servlet、Filter 和 Listener. 除此之外, 其他的 Servlet、Filter 和 Listener 全部拆分到各个模块, 在 web.xml 片段中进行配置. 全局 Servlet、Filter 和 Listener 各自的作用如下:

全局 Servlet: 负责拦截所有请求, 然后把请求转交给一个全局 Servlet 代理, 再由全局 Servlet 代理把请求分发到各个模块中的 Servlet. 全局 Servlet 代理维护着一个资源注册器的键-值对集合, 资源注册器分为 Servlet 注册器和 Resource 注册器两种类型[1], 分别用于处理 Servlet 请求和静态资源请求. Servlet 注册器以模块上下文路径拼接 servlet-mapping 作为键, 例如 /moduleContextPath/login/*; Resource 注册器以模块上下文路径拼接静态资源文件后缀作为键, 例如 /moduleContextPath/*.html.

全局 Filter: 为模块中 Filter 提供一个过滤请求的入口, 将各模块的 Filter 组装成 FilterChain, 然后依次对请求进行过滤处理.

全局 Listener: Web 应用中的事件监听器可分为 Servlet 上下文(ServletContext)事件监听器类、HTTP 会话 (HttpSession) 事件监听器类以及 Servlet 请求 (ServletRequest)事件监听器类等 3 种类型[1]. 可在 Web 应用的 web.xml 中配置相应类型 Listener 的全局代理, 用于捕获相关事件, 然后将事件传递给模块内部相同类型的 Listener.

Web 应用启动时, 会首先按传统方式初始化配置在 web.xml 中的全局 Servlet、Filter 和 Listener. 在初始化全局 Servlet 时, 通过反射执行 OSGi 的实现框架的启动方法[4](例如 Equinox 的 org.eclipse.core.runtime.adaptor.EclipseStarter 类的 startup 方法), 启动 OSGi 的框架, 再由 OSGi 框架依次启动每个模块. 模块的启动包括以下过程:

(1) 调用模块激活器的启动方法. OSGi 规范规定模块可以定义一个激活器, 激活器中包括 start 和 stop 方法, 这两个方法将分别在模块启动和停止时被调用. 模块启动过程中首先会调用模块激活器的 start 方法, 可以把模块启动之前要执行的操作放在此方法中.

(2) 解析 web.xml 片段. 在模块启动的过程中特定目录进行扫描, 并对发现的 web.xml 片段文件解析, 处理模块中配置的 Servlet、Filter 和 Listener.

Servlet: 对 Servlet 进行封装, 封装的内容包括模块上下文路径和模块类加载器, 然后以模块上下文路径拼接 servlet-mapping 后缀为键, 以封装的 Servlet 对象为值, 注册到全局 Servlet 代理中.

Filter: 创建并初始化 Filter 对象, 然后将 Filter 添加到全局 Filter 中, 形成 FilterChain, 对请求依次进行过滤处理.

Listener: 对于除了 ServletContextListener 之外的 Web 应用事件监听器, 将创建并初始化 Listener 对象, 然后添加到对应类型的全局 Listener 中, 接收全局 Listener 传递过来的相关事件, 并触发相应的处理; ServletContextListener 则需要进行特殊处理. ServletContextListener 是对 Servlet 上下文初始化和销毁事件的监听, Web 应用模块化后, 会先启动应用然后再逐个启动模块, 模块启动时 Web 应用的 Servlet 上下文已经被初始化过了, 模块中的 ServletContextListener 无法捕获到 Servlet 上下文初始化事件, contextInitialized 方法也不会被动触发. 因此需要对此做出补偿, 在模块启动过程中主动执行 ServletContextListener 的 contextInitialized 方法. 同样的, 在模块停止时, 需要提前主动执行 ServletContextListener 的 contextDestroyed 方法.

(3) 处理 JSP 页面. 把 JSP 作为 JspServlet 进行处理, 为每个模块创建一个 JspServlet, 并对 JspServlet 进行封装, 封装的内容包括模块上下文路径和模块类加载器, 然后以模块上下文路径拼接 JSP 文件后缀为键(例如/moduleContextPath/*.jsp), 以封装的 JspServlet 对象为值, 注册到全局 Servlet 代理中.

(4) 注册静态资源文件. 在模块启动的过程中, 扫描模块中的静态资源文件, 然后按文件类型, 以模块上下文路径拼接文件后缀为键, 以封装了模块对象的资源对象为值, 注册到全局 Servlet 代理中.

模块化的 Web 应用启动后的内部结构如图 5 所示, 模块中的 Filter 和 Listener 都被组装到应用的全局 Filter 和 Listener, 模块中 Servlet、JSP 和静态资源文件也被注册到全局 Servlet 代理中, 为模块化环境下的请求处理做好了准备.

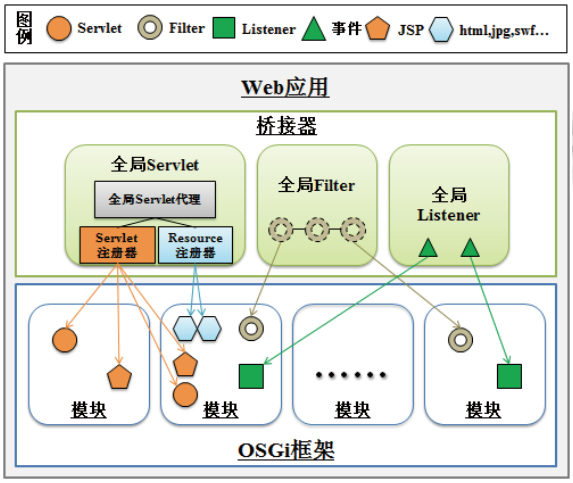

图 5　模块化的 Web 应用内部结构

2.5 请求处理

模块化 Web 应用的请求处理过程如图 6 所示, 具体包括以下过程:

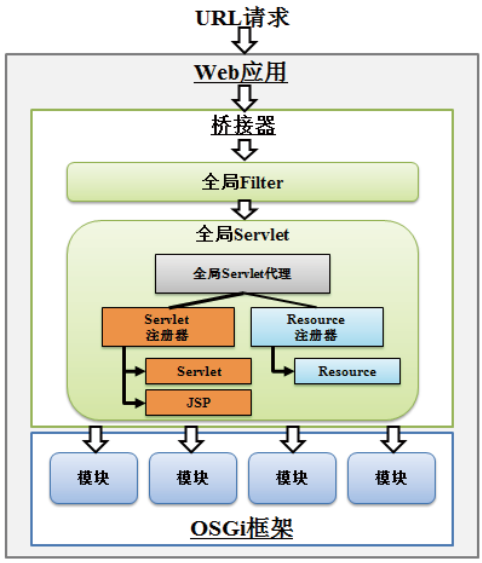

图 6　请求处理过程示意图

(1) 应用服务器接收到用户的 URL 请求后, 把请求交给部署在服务器上的 Web 应用处理.

(2) 配置在 Web 应用 web.xml 中的全局 Filter 会拦截所有请求, 然后把请求交给各模块 Filter 组成的 FilterChain, 由各模块 Filter 依次进行请求过滤处理.

(3) 配置在 Web 应用 web.xml 中的全局 Servlet 接收所有请求, 然后把请求转交给全局 Servlet 代理. Web 应用启动的过程中, 各模块已经将本模块的 Servlet、JSP 和静态资源文件注册到了全局 Servlet 代理中. 全局 Servlet 代理接收到请求后, 对 URL 进行解析, 解析规则就是根据 URL 中模块上下文路径及之后的部分找到能处理该请求的 Servlet 注册器或者 Resource 注册器, 例如 http://localhost:9000/webapp/workbench/login.jsp 将被转交给键为 workbench/*.jsp 的 Servlet 注册器处理. 如果无法找到匹配的注册器, 则说明模块中无法处理该请求, 直接返回请求异常.

(4) 资源注册器本身并不对请求进行处理, 它会继续把请求交给内部的 Servlet 和 Resource 对象处理.

Servlet 注册器会把请求转给封装了模块上下文对象的 Servlet 对象, Servlet 对象处理请求时, 会首先把当前线程的类加载器切换成当前模块的类加载器, 这样 Servlet 对象才能遵从 OSGi 规范的类加载委托模型, 与本模块或依赖的其他模块中的类和资源文件进行交互, 完成对请求的处理. 请求处理完成后, 再恢复当前线程的类加载器.

对于 JSP 这种特殊的 Servlet, Servlet 对象还需要额外完成: 1)从模块的目录结构中查找并读取被请求的 JSP 文件; 2)对 JSP 文件进行编译; 3)加载编译后的类文件; 4)调用类方法等一系列处理. 对于传统 Web 应用来说, 这些处理通常是由应用服务器来完成的[5], Web 应用模块化后, 因为 JSP 文件被从 Web 应用的固定目录转移到了模块内部目录下, 对应用服务器变成了不可见, 所以需要自行完成对 JSP 的处理.

Resource 注册器会把请求转给封装了模块上下文对象(包括模块上下文路径、模块类加载器)的 Resource 对象, Resource 对象可利用模块上下文对象从模块的目录结构中查找并读取被请求的资源文件, 然后返回请求.

3　应用案例

为了充分发挥 Web 应用模块化架构的特性, 我公司实现了一个 Web 应用开发平台, 功能全部由模块组

成. 该平台采用微内核模式, 将功能分为核心功能、基本功能和可选功能. 核心功能负责实现 Web 应用模块化架构, 基本功能提供通用开发框架. 所有应用系统基于平台进行开发, 不同应用系统可根据需求选择可选功能中的模块, 同一个应用系统也可以将功能分成通用和个性化两部分. 图 7 为我公司基于平台开发的电力营销系统功能组件图, 同一套应用系统部署在北京和天津这两个不同的地点时, 把平台核心功能、基本功能、可选功能、通用业务功能和相应的个性化业务功能模块组合起来即可.

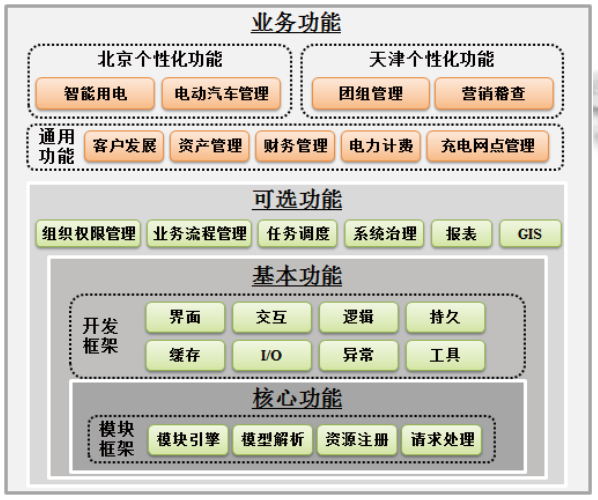

图 7 应用系统功能组件图

4 结语

为了解决 Web 应用在模块化和动态性方面的不足, 本文结合当前 Web 应用的关键技术点 Servlet、JSP、静态资源、请求处理等, 基于 OSGi 技术给出了 Web 应用的模块化架构. 通过我公司应用系统的实践证明该架构将在不同的开发阶段为 Web 应用带来模块化能力和动态性支持:

(1) 设计阶段: 模块的物理隔离使得“模块化”设计思想、“高内聚低耦合”设计原则从技术上得到落实和保障;

(2) 开发阶段: 开发人员可以只关注自己负责的模块的代码而不是整个应用的代码, 不用被不相关模块的问题所干扰, 实现关注分离;

(3) 集成阶段: 应用的所有功能都由模块提供, 应用集成就是模块的简单叠加. 各个模块之间结构清晰, 集成过程出现问题时, 也很容易定位到具体的模块, 节省解决问题的时间;

(4) 运行阶段: 可动态的安装、启动、更新、刷新、停止和卸载模块, 实现应用运行时的动态更新.

参考文献

1 OSGi Alliance. OSGi Enterprise Release 5, 2012. 29–44, 447–460.

2 Java Community Process. JSR-315, Java Servlet Specification v3.0, 2009. 97–114.

3 OSGi Alliance. OSGi Core Release 5, 2012. 23–112.

4 林昊,曾宪杰.OSGi 原理与最佳实践.北京:电子工业出版社, 2009.7–37.

5 Java Community Process. JSR-245, Java Server Pages Specification v2.2, 2009. 3–16.

(上接第 72 页)

参考文献

1 刘芳华.基于 ARM 的 WiFi 无线通信终端的研究与实现.武汉:武汉科技大学,2010.

2 万书芹,魏斌,陈子逢,等.一种无线传感网网关的设计.电子与封装,2012,12(4):38–41,48.

3 陈琦,韩冰,秦伟俊,等.基于 Zigbee/GPRS 物联网网关系统的设计与实现.计算机研究与发展,2011,48(S):367–372.

4 刘涛,王洋.一种基于 ARM 和 GPRS 的远程抄表集中器的设计.机械管理开发,2011,(5):24–26.

5 张闯.基于 ARM/GPRS 的远程水文监测终端设计[学位论文].哈尔滨:哈尔滨理工大学,2010.

6 陈易厅,陈安,胡跃明.LED 生产设备智能信息化平台设计.自动化与仪表,2011,(9):23–26.

7 王宝珂.基于 ARM11 的嵌入式视频监控系统终端的设计.南京:南京理工大学,2012.

8 李颀,张小件.基于 PXA322 嵌入式 GPRS 数据采集与传输系统的设计.仪表技术与传感器,2013,(3):69–71,75.