Sun Java™ System

Content Delivery Server 5.1
定制指南

Sun Microsystems, Inc.
www.sun.com

文件号码：820-5375-10
2008 年 6 月

请将有关本文档的意见和建议提交至：http://www.sun.com/sunsurveys/dsc/dsc-feedback.jsp

---

请回收

Adobe PostScript

目录

图

表

代码样例

前言

《Sun Java™ System Content Delivery Server 5.1 定制指南》对每个 Content Delivery Server 应用程序编程接口 (Application Programming Interface, API) 进行了说明。这些 API 用于将 Content Delivery Server 与现有基础架构集成在一起。订户 API 和 Vending Manager API 用于访问 Content Delivery Server 数据。

阅读本书之前

本指南适用于负责基于 API 编写适配器的程序员以及负责将 Content Delivery Server 与现有基础架构集成的系统管理员。它假定管理员在 Java 编程语言和联网、数据库及 Web 技术方面具有一定的知识。

本文档的组织结构

- 第 1 章对 Content Delivery Server API 进行了概述。
- 第 2 章介绍了事件服务 API，该 API 提供事件的异步报告。
- 第 3 章介绍了记帐 API，该 API 提供 Content Delivery Server 与记帐系统之间的接口。
- 第 4 章介绍了内容管理 API，该 API 用于管理 Content Delivery Server 与内容管理系统之间的接口。
- 第 5 章介绍了内容验证 API，该 API 用于验证和保护提交给 Content Delivery Server 的内容。
- 第 6 章介绍了用户配置 API，该 API 用于在系统中添加、删除、检索、更新、启用和禁用用户。

- 第 7 章介绍了 WAP 网关 API，该 API 用于从 HTTP 头中检索 MSISDN、设备配置文件和其他属性。
- 第 8 章介绍了消息传送 API，该 API 提供了一种机制，使运营商或应用程序供应商能够集成自身的无线访问协议 (Wireless Access Protocol, WAP)、短消息服务 (Short Message Service, SMS) 和多媒体消息服务 (Multimedia Messaging Service, MMS) 推送实现。
- 第 9 章介绍了 Content Delivery Server 如何处理从多媒体消息服务中心 (Multimedia Messaging Service Center, MMSC) 或其他外部实体发送的确认消息。
- 第 10 章介绍了流式传输 API，该 API 用于与流式传输服务器进行通信以管理流式传输的内容。
- 第 11 章介绍了订户 API，通过该 API 可以访问由 Content Delivery Server 维护的数据。
- 第 12 章介绍了 Vending Manager API，外部应用程序可以通过该 API 访问包、活动和订户计划。
- 第 13 章介绍了设备客户机 Web 服务，通过该服务可以远程访问订户 API 以及 Content Delivery Server 使用 Web 服务维护的数据。

Shell 提示符

Shell	提示
C shell	*machine-name*%
C shell 超级用户	*machine-name*#
Bourne shell 和 Korn shell	$
Bourne shell 和 Korn shell 超级用户	#

印刷约定

字体	含义	示例
AaBbCc123	命令、文件和目录的名称；计算机屏幕输出	编辑 .login 文件。 使用 ls -a 列出所有文件。 % You have mail.
AaBbCc123	用户键入的内容，与计算机屏幕输出的显示不同	% **su** Password:
AaBbCc123	保留未译的新词或术语以及要强调的词。要使用实名或值替换的命令行变量。	这些称为 *class* 选项。 要删除文件，请键入 **rm** *filename*。
新词术语强调	新词或术语以及要强调的词。	您**必须**成为超级用户才能执行此操作。
《书名》	书名	阅读 《用户指南》的第 6 章。

注 – 字符的显示方式随浏览器设置的不同而有所不同。如果未正确显示字符，请在浏览器中将字符编码更改为 Unicode UTF-8。

相关文档

下表列出了本产品的文档。可以从以下位置获取联机文档：

http://docs.sun.com/app/docs/prod/cds

应用	书名	文件号码	格式	位置
门户署名和本地化	《Sun Java™ System Content Delivery Server 5.1 署名和本地化指南》	820-5374-10	PDF HTML	$CDS_HOME/Documentation/branding 和联机文档。
规划系统	《Sun Java™ System Content Delivery Server 5.1 容量规划指南》	820-1939-10	PDF HTML	$CDS_HOME/Documentation/capacity 和联机文档。
使用 Developer Portal	《Sun Java™ System Content Delivery Server 5.1 内容开发者指南》	820-5376-10	PDF HTML	$CDS_HOME/Documentation/devguide 和联机文档。

应用	书名	文件号码	格式	位置
使用提供的 API	《Sun Java™ System Content Delivery Server 5.1 定制指南》	820-5375-10	PDF HTML	`$CDS_HOME/Documentation/customization` 和联机文档。
疑难解答	《Sun Java™ System Content Delivery Server 5.1 错误消息》	820-5377-10	PDF HTML	`$CDS_HOME/Documentation/errmsgs` 和联机文档。
安装系统	《Sun Java™ System Content Delivery Server 5.1 安装指南》	820-5378-10	PDF HTML	`$CDS_HOME/Documentation/install` 和联机文档。
设置和集成现有的基础架构	《Sun Java™ System Content Delivery Server 5.1 集成和配置指南》	820-5379-10	PDF HTML	`$CDS_HOME/Documentation/integration` 和联机文档。
迁移到最新发行版本	《Sun Java™ System Content Delivery Server 5.1 迁移指南》	820-1945-10	PDF HTML	`$CDS_HOME/Documentation/migration` 和联机文档。
产品参考信息	《Sun Java™ System Content Delivery Server 5.1 参考手册》	820-5380-10	PDF HTML	`$CDS_HOME/Documentation/refman` 和联机文档。
监视和管理系统	《Sun Java™ System Content Delivery Server 5.1 系统管理指南》	820-5381-10	PDF HTML	`$CDS_HOME/Documentation/system-mgmt` 和联机文档。

文档、支持和培训

Sun 服务	URL
文档	`http://www.sun.com/documentation/`
支持	`http://www.sun.com/support/`
培训	`http://www.sun.com/training/`

第三方 Web 站点

Sun 对本文档中提到的第三方 Web 站点的可用性不承担任何责任。对于此类站点或资源中的 （或通过它们获得的）任何内容、广告、产品或其他资料， Sun 并不表示认可，也不承担任何责任。对于因使用或依靠此类站点或资源中的 （或通过它们获得的）任何内容、产品或服务而造成的或连带产生的实际或名义损害或损失， Sun 不承担任何责任。

Sun 欢迎您提出意见

Sun 致力于提高其文档的质量，并十分乐意收到您的意见和建议。可通过以下方式提交您的意见：访问 `http://docs.sun.com`，然后单击 “发送意见” (Send Comments)。

请在反馈中包括文档的标题和文件号码：

《Sun Java™ System Content Delivery Server 5.1 定制指南》，文件号码 820-5375-10。

第 1 章

简介

要部署商业服务，运营商必须将 Sun Java System Content Delivery Server 与现有基础架构集成在一起。记帐系统、用户管理系统和报告系统都具有集成要求，并且涉及运营商的复杂级别。 Sun Java System Content Delivery Server API 对此系统集成很有帮助。

本节介绍了下列 API：

- 事件服务 API
- 记帐 API
- 内容管理 API
- 内容验证 API
- 用户配置 API
- WAP 网关 API
- 消息传送 API
- 确认服务 API
- 流式传输 API
- 订户 API
- Vending Manager API
- 设备客户机 Web 服务

第 1-5 页上的第 1.13 节 “按功能标识的 API”按功能标识用于将 Content Delivery Server 与现有系统集成的 API。

1.1 事件服务 API

事件服务 API 提供事件的异步报告，以便外部系统能够提取这些事件或收到事件通知。事件服务 API 使用由管理器组件发布的所有消息。对于每条消息，将提取消息的上下文和详细信息，将其插入数据库中，并进行传播以便在记帐、报告、消息传送和广告中使用。

1.2 记帐 API

记帐 API 支持预付和同步记帐模型。对于支持在购买内容之前收费的记帐系统，记帐 API 用于在允许订户下载内容之前验证其帐户资金足够。对于要求实时记帐的记帐系统，记帐 API 用于在内容被购买时对订户帐户进行收费。

通过记帐 API，运营商还可以对不是在 Vending Manager 中指定的内容收取不同的费用。例如，如果运营商要为商务订户提供特殊折扣，记帐 API 将用于从记帐系统中检索特殊价格并将该价格显示给客户。

1.3 内容管理 API

通过内容管理 API 可以在内容下载到订户设备时对其进行测试。在传给订户之前需要使用用户特定数据或记帐特定数据立即测试内容时使用此 API。内容管理 API 还可以用于更改内容的属性，例如 MIME 类型或内容类型。

1.4 内容验证 API

内容验证 API 处理提交给 Content Delivery Server 的内容。使用此 API 可创建自动验证提交内容时所需的内容验证适配器。在将内容提交到 Content Delivery Server 时执行的提交验证器工作流会使用内容验证适配器。

1.5 用户配置 API

用户配置 API 提供了可进入现有订户数据库的接口。它集成了订户、应用程序开发者、管理员和设备的数据源。运营商无需创建单独的新数据库即可使用 Content Delivery Server。如果运营商需要组合或合并多个数据库项，则用户配置 API 将有助于集成此数据，而不会影响其他服务请求信息。它还有助于集成旧版信息。

通过用户配置 API， Content Delivery Server 为所有访问数据库的组件提供了一个公共服务层。将数据访问抽象化使得 Content Delivery Server 可以不受特定数据库的束缚。通过对请求中频繁使用的数据加以缓存，此服务提供了对基本数据的可伸缩访问。与安全机制结合使用，可以跨不同的用户或管理员控制数据访问。

1.6 WAP 网关 API

WAP 网关 API 用于处理手机识别号 (MSISDN) 验证与头传输中的实现差异。使用 MSISDN 进行验证和会话管理为订户和运营商提供了便利。 Content Delivery Server 支持单点登录，因此，如果订户通过移动电话访问系统，则可以使用请求标题执行验证，而无需提供订户的用户名和密码。 Content Delivery Server 可以配置为与多个网关类型并行通信。

1.7 消息传送 API

消息传送 API 提供了一种机制，使运营商或应用程序供应商可以通过提供适配器来集成自身的 WAP 或短消息服务 (Short Message Service, SMS) 推送实现。Content Delivery Server 还提供了可以在大多数情况下使用的默认 WAP 和 SMS 推送实现。

1.8 确认服务 API

通过确认服务 API， Content Delivery Server 可以处理多媒体消息服务中心 (Multimedia Messaging Service Center, MMSC) 发送的确认消息。确认信息一般在内容下载到设备后发送。此 API 用于创建指向 MMSC 的连接并监视 MMSC 发送的消息。

1.9 流式传输 API

流式传输 API 处理 Content Delivery Server 与流式传输服务器之间的交互。如果 Vending Manager 储存按需流式传输的内容，则会将其复制到流式传输服务器。如果 Vending Manager 不储存该内容，则会将其从流式传输服务器中删除。当订户购买内容时，将提供用于启动流的 URL。

流式传输 API 包含一些类，用于描述流式传输的内容和购买内容的订户。如果系统支持流式传输的内容，则可以使用该 API 来创建流式传输适配器。

1.10 订户 API

通过订户 API 可以访问由 Content Delivery Server 维护的数据。使用此 API 可以获取创建客户机应用程序所需的数据，以便订户可以访问由 Content Delivery Server 管理的内容。订户 API 可以由基于 Java 技术的本地应用程序（Java 应用程序）直接访问，也可以使用 XML-RPC 从远程应用程序或使用 Java 之外的编程语言编写的应用程序进行访问。

1.11 Vending Manager API

可以使用 Vending Manager API 访问由 Content Delivery Server 维护的 Vending Manager 数据。可以使用该 API 创建和管理包、活动、订户群和订户计划，而无需通过 Vending Manager 管理控制台界面来实现。可以通过 XML-RPC 来访问 Vending Manager API。

1.12 设备客户机 Web 服务

可以使用设备客户机 Web 服务通过 Internet 访问订户 API。可以使用这些 Web 服务通过标准 Web 协议访问由 Content Delivery Server 维护的数据。

1.13 按功能标识的 API

下表表明了用于将某些 Content Delivery Server 功能与现有系统集成的 API。要获得想要在所安装的 Content Delivery Server 中支持的功能，必须实现指定的 API。

表 1-1 到 Content Delivery Server API 的功能映射

功能	系统	API
为朋友购买	简单邮件传输协议 (Simple Mail Transfer Protocol, SMTP)、短消息服务中心 (Short Message Service Center, SMSC)、推送代理网关 (Push Proxy Gateway, PPG) 和多媒体消息服务中心 (Multimedia Message Service Center, MMSC)	消息传送 API
告诉朋友	SMTP、 SMSC、 PPG 和 MMSC	消息传送 API
事件驱动的活动*	SMTP、 SMSC、 PPG 和 MMSC	消息传送 API
外发活动*	SMTP、 SMSC、 PPG 和 MMSC	消息传送 API
确认推送消息	SMTP、 SMSC、 PPG 和 MMSC（服务必须返回确认消息）	确认服务 API
外部用户数据库	轻量目录访问协议 (Lightweight Directory Access Protocol, LDAP)	用户配置 API
记帐 - 预付	预付记帐系统	记帐 API
记帐 - 后付费	后付费记帐系统	事件服务 API
与外部数字权限管理 (External Digital Rights Management, DRM) 集成	外部 DRM 引擎	内容验证 API, 内容管理 API
单点登录和设备登录	WAP 网关（必须传递 MSIDN 或唯一 ID）	WAP 网关 API
流式传输	流式传输服务器	流式传输 API
门户集成	Web 门户	订户 API, 设备客户机 Web 服务
包、活动、订户计划、订户群	Vending Manager	Vending Manager API

* 促销活动不要求与外部系统集成

第 2 章

事件服务 API

本章介绍了 Sun Java System Content Delivery Server 事件服务 API。事件服务 API 包含以下外部接口：

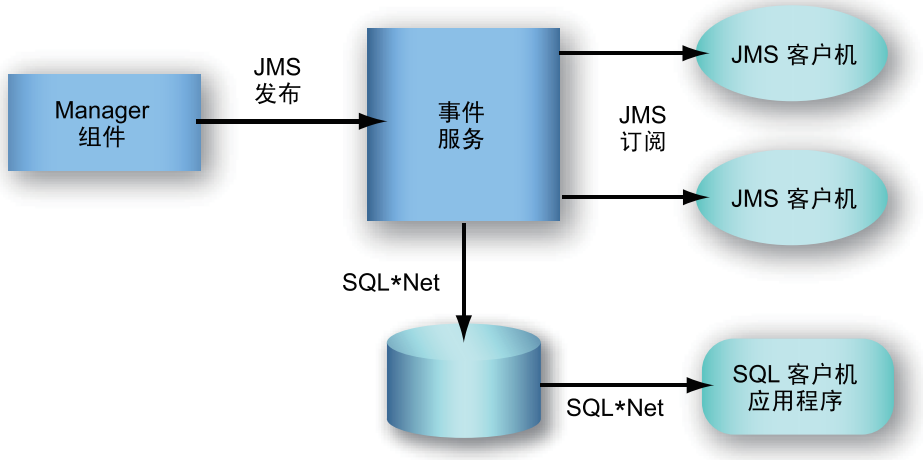
- 用于通过数据库客户机应用程序直接查询事件数据的数据库结构
- 可以由 JMS 客户机应用程序订阅的 Java 消息服务 (Java Message Service, JMS) 主题

事件服务将来自事件队列的消息传播到已订阅了事件服务所发布主题的任何相关事件监听器。事件服务还将事件数据存储在 Content Delivery Server 数据库中。可以使用数据库客户机或 JMS 客户机应用程序方法。这两种方法都可以近乎实时地访问相同的信息。

使用事件服务 API 可编写应用程序，以监听特定事件并根据企业需要采取操作。例如，可以使用此 API 编写定制的记帐适配器，以处理购买事件并在购买内容后向订户收费。

图 2-1 简单说明了与事件服务进行交互的通用系统和组件，以图解方式介绍了信息是如何通过各种系统交互进行传递的。

图 2-1 事件服务概述



Content Delivery Server Catalog Manager 和 Vending Manager 组件将相应的消息发布到 JMS 队列中。这些消息由事件服务进行检索和处理。Content Delivery Server 中的任何组件都可以将事件发布到事件服务。由于事件发布是异步操作，因此一旦发送消息，发布消息的组件将继续操作。

事件服务在 Content Delivery Server 环境中作为单独的进程运行。事件服务执行以下任务：

- 使用由各种服务器组件发布的所有消息
- 提取消息的上下文和详细信息，并将该信息插入到数据库中
- 在成功处理消息后向 JMS 队列发送确认
- 出现错误时将消息放入错误队列中

  可以随后处理消息，并可能将其重新提交到事件队列中。消息仍确认为由事件服务成功处理。

事件服务 API 的当前实现使用 JMS 点对点 (Point-to-Point, PTP) 消息传送域，从中各种发布程序将消息发布到单个队列，并且事件服务 JMS 客户机应用程序实例将处理这些消息。

事件服务成功处理完事件后，这些事件将被放入数据库中。当事件服务与某个 Vending Manager 一起部署时，这些事件将被放入 Vending Manager 数据库结构中。当事件服务仅与某个 Catalog Manager 一起部署时，这些事件将被放入 Catalog Manager 数据库结构中。有关存储这些消息的数据模型信息，请参见第 2-3 页上的第 2.1.1 节 “事件表”。

2.1 数据库客户机应用程序

数据驻留在标准关系数据库中，开发者可以通过一组视图以任何能够查询 Oracle 数据库的编程语言编写应用程序。例如，可以使用 Java 编程语言、C++ 或 Visual Basic 编写这些应用程序。

2.1.1 事件表

本节所介绍的表格包含事件服务处理完事件后产生的事件数据。作为集成者，您具有访问这些表格数据的权限。

- 如果将事件服务与某个 Vending Manager 一起部署，请使用用户 *prefix*_vs_app 连接到 Oracle 数据库。其中 *prefix* 是为数据库配置文件（用于创建数据库）中 Vending 元素下的 Prefix 元素指定的值。请使用为 Vending 元素下的 Password 元素指定的密码。
- 如果仅将事件服务与某个 Catalog Manager 一起部署，请使用用户 *prefix*_ps_app 连接到 Oracle 数据库。其中 *prefix* 是为数据库配置文件（用于创建数据库）中 Catalog 元素下的 Prefix 元素指定的值。请使用为 Catalog 元素下的 Password 元素指定的密码。

有关数据库配置文件的信息，请参见《Sun Java™ System Content Delivery Server 5.1 安装指南》。

2.1.1.1 CDS_EVENT 表

CDS_EVENT 表包含事件服务成功处理的每个事件的记录。处理事件时，此表将实时更新。

表 2-1 CDS_EVENT 表

列名	数据类型	描述
CDS_EVENT_ID	NUMBER(18)	系统生成的用作记录 ID 的唯一数字。此字段是表的主键。
CDS_EVENT_DATE	DATE	表示事件消息生成时间的时间戳。
CDS_EVENT_TYPE_ID	NUMBER(18)	CDS_EVENT_TYPE 表的外键。
EVENT_SOURCE_ID	NUMBER(18)	EVENT_SOURCE_TYPE 表的外键。
SVR_INSTANCE_ID	NUMBER(18)	系统数据。
SVR_SESSION_ID	VARCHAR2(128)	系统数据。

表 2-1 CDS_EVENT 表 （续）

列名	数据类型	描述
SUB_SYSTEM_ID	VARCHAR2(80)	与会话及事件关联的 MSISDN。如果 MSISDN 未知，则该列为 Null。
CDS_USER_ID	NUMBER(18)	与会话及事件关联的 Content Delivery Server 用户 ID。如果用户未登录，则此列可能为 Null。
VENDOR_ID	NUMBER(18)	内容提供商 ID。
CONTENT_ID	NUMBER(18)	内容表的外键。
RAW_EVENT_MESSAGE	CLOB	原始事件消息 XML。
CREATE_DATE	DATE	系统数据。
MOD_DATE	DATE	系统数据。
LOCK_VERSION	NUMBER(1)	系统数据。

2.1.1.2 CDS_EVENT_TYPE 表

CDS_EVENT_TYPE 表是一个包含事件类型定义的静态表。在第 2-6 页上的第 2.3 节“事件和事件数据”中列出了这些定义。

表 2-2 CDS_EVENT_TYPE 表

列名	数据类型	描述
CDS_EVENT_TYPE_ID	NUMBER(18)	系统生成的用作记录 ID 的唯一数字。此字段是表的主键。
CDS_EVENT_GROUP_ID	NUMBER(18)	EVENT_GROUP 表的外键。用于对事件类型进行分组。
CDS_EVENT_TYPE_NAME	VARCHAR2(80)	可读名称。
DESCRIPTION	VARCHAR2(80)	特定事件类型的描述。
LONG_DESCRIPTION	VARCHAR2(255)	事件类型的详细描述 （如果需要）。
IS_ACTIVE	NUMBER(1)	表示是否为活动事件类型的标志。
CREATE_DATE	DATE	系统数据。
MOD_DATE	DATE	系统数据。
LOCK_VERSION	NUMBER(1)	系统数据。

2.1.1.3 EVENT_SOURCE_TYPE_ID 表

EVENT_SOURCE_TYPE_ID 表是一个包含事件源类型定义的静态表。

表 2-3 EVENT_SOURCE_TYPE_ID 表

列名	数据类型	描述
EVENT_SOURCE_TYPE_ID	NUMBER(18)	系统生成的用作记录 ID 的唯一数字。此字段是表的主键。
EVENT_SOURCE_TITLE	VARCHAR2(255)	可读名称。
CREATE_DATE	DATE	系统数据。
MOD_DATE	DATE	系统数据。
LOCK_VERSION	NUMBER(1)	系统数据。

2.1.2 报告工具

可以使用任何连接到 Oracle 数据库的数据库报告工具，基于此数据生成各种类型的报告，如 Crystal Reports 或 Jasper Reports。有关报告数据的其他信息，请参见《Sun Java™ System Content Delivery Server 5.1 集成和配置指南》中的第 1.5 节“设置定制报告”。

2.2 JMS 客户机应用程序

除了基于 SQL 的事件数据接口，还可以实现通过 JMS 主题与事件服务直接相连的 JMS 客户机应用程序。尽管此方法比较难于实现，但它在使用由 Content Delivery Server 生成的事件方面具有更大的灵活性。

要成功使用此 API 与事件服务集成，您需要熟悉如何编写 JMS 客户机应用程序，并且了解在 Java 2 Platform Enterprise Edition（J2EE™ 平台）的 JMS 规范中介绍的发布/订阅消息传送域。任意数量的 JMS 客户机应用程序都可以使用发布/订阅消息传送模型来订阅由事件服务发布的消息。

2.3 事件和事件数据

本节介绍了有关由事件服务提供的事件和事件数据的信息。数据库和 JMS 客户机应用程序都使用此信息过滤和处理事件。客户机应用程序可以根据企业需要处理各种类型的事件。例如，在收到 sms_content_push_sent 事件时，可以执行收费短消息 (premium SMS) 记帐操作。

下表介绍了由事件服务生成的事件。

表 2-4 事件

事件	描述
content_changed	Catalog Manager 管理员对内容进行了更改。
content_purchased	订户购买了某个内容项或下载了免费的内容项。
content_refunded	对某个内容项发放了退款。
download_deleted	从设备中删除了下载的内容。
download_error	设备指明下载内容时出错。
download_initiated	订户已开始下载内容。
download_install_notified	设备已确认下载成功。
external_content_updated	外部托管内容已更新。
gift_cancelled	礼品订阅被取消。
gift_download_confirm	接收者下载了礼品。
gift_download_deleted	接收者下载的礼品被删除。
gift_download_error	下载礼品时出错。
gift_download_initiated	礼品接收者已开始下载礼品。
gift_expired	礼品已过期。
gift_purchased	作为礼品购买了某个内容项。
gift_refunded	对订户购买的礼品发放了退款。
gift_subscription_purchased	作为礼品购买了某个内容项的订阅。
gift_usage_purchased	作为礼品购买了对某个内容项的多次使用权。
mms_push_sent	MMS 消息已发送。
pricing_changed	已使用 “类别价格编辑” 功能对一个或多个内容项的价格进行了更改。
sms_content_push_sent	在 SMS 消息中发送了内容二进制代码。请使用此事件来触发收费短消息 (premium SMS) 记帐。

表 2-4 事件 （续）

事件	描述
sms_push_sent	SMS 消息已发送。
sms_received	SMS 消息已收到。
smtp_push_sent	SMTP 消息已发送。
status_changed_to_deleted	某个内容项的状态被更改为 “已删除”。
status_changed_to_denied	某个内容项的状态被更改为 “已拒绝”。
status_changed_to_new	某个内容项的状态被更改为 “新建”。
status_changed_to_pending	某个内容项的状态被更改为 “待定”。
status_changed_to_published	某个内容项的状态被更改为 “已发布”。
status_changed_to_testing	某个内容项的状态被更改为 “测试”。
submission_failed	提交的内容被 Content Delivery Server 拒绝。
submission_successful	提交的内容被 Content Delivery Server 接受。
subscriber_registered	订户已成功注册。
subscription_cancelled	对某个内容项的订阅被取消。
subscription_purchased	对某个内容项的订阅被购买。
usage_purchased	已购买了对某个内容项的多次使用权。
validation_passed	提交的内容由提交验证器工作流成功处理。
validation_failed	提交的内容在提交验证器工作流的某个步骤中失败。
wap_push_sent	WAP 消息已发送。

下表列出了可以包含在事件中的信息。每个事件仅包含与该事件相关的参数。

表 2-5 事件数据

参数	数据类型	描述
billing-ticket	字符串	该事务的记帐凭单。
campaign_coupon	字符串	活动的礼券代号。
campaign_id	字符串	活动的唯一标识符。
catalog-res-id	字符串	内容版的唯一标识符。
content_binary_mimetype	字符串	内容的 MIME 类型。
content_class_id	字符串	内容项的唯一标识符。
content_description	字符串	内容的详细描述。
content_drm_type_id	字符串	标识用于保护内容的 DRM 方法的字符串。

表 2-5 事件数据 （续）

参数	数据类型	描述
content_short_description	字符串	内容的简短描述。
content-id	字符串	所购买的内容的唯一标识符。此值与 catalog-res-id 相同。
content_name	字符串	内容的名称。
current-status	字符串	事务的当前状态。
date	日期	事务发生的日期。
destination-address	字符串	向其发送内容的地址，例如，请求内容的订户的 MSISDN。
developer-content-id	字符串	开发者用于标识内容的唯一标识符。
developer-id	字符串	内容开发者的唯一标识符。
developer_name	字符串	提交内容的开发者的姓名。
download-confirm	布尔型	表示成功下载后是否需要确认的标志。
download-count	整型	根据支付的价格确定的内容下载次数。
download-current-count	整型	订户下载此内容的次数 （包括本次）。
download-expiration	布尔型	表示下载时段是否已过期的标志。
download-period	整型	允许下载内容而无需向订户收取其他费用的时段。
download-price	浮点型	购买内容的价格。
download-purchase	布尔型	用于表示购买请求的标志。
download-recurring	布尔型	表示订户是否为每次下载付费的标志。
event-log	字符串	事件日志的名称。
event-msg	字符串	随事件一起发出的消息。
event-source-type-id	整型	用于标识事件源的编号。
event-type	整型	发生的事件的数字表示。
event-type-id	字符串	发生的事件类型。
external_content_id	字符串	记帐系统用于标识内容的标记。
external_group_id	字符串	记帐系统用于标识内容所属组的标记。
external-request-text	字符串	订户的请求文本，例如， MO 推送请求内容。
gift_message	字符串	礼品中包含的消息。
gifted_current_downloads	整型	接收者下载此礼品的次数 （包括本次）。
gifted_current_subscriptions	整型	接收者使用的订阅时段数 （包括本时段）。
gift_download_date	日期	接收者第一次下载礼品的日期。

表 2-5 事件数据（续）

参数	数据类型	描述
gift_expiration_date	日期	接收者认领礼品的最后期限。
gift_purchase_date	日期	送礼人购买礼品的日期。
gifted_downloads	整型	礼品中包含的下载次数。
gifted_subscriptions	整型	礼品中包含的订阅时段数。
is_on_device	布尔型	表示内容是否已存在于设备中的标志。
is-prepay	布尔型	表示订户是否已为内容预付费的标志。
locale	字符串	订户的语言环境。
MSISDN	字符串	订户设备的 MSISDN。
pricingoption_key	字符串	用于标识价格选项的字符串。
pricingoption_name	字符串	价格选项的名称。
push-msgtext	字符串	发送到订户设备的消息或电子邮件。
recipient_locale_code	字符串	内容预定接收者的语言环境。
recipient_login_id	字符串	内容预定接收者的登录 ID。
recipient_mobile_id	字符串	内容预定接收者的移动 ID。
recipient_unique_device_id	字符串	预定接收者的唯一设备 ID。
server-id	字符串	Vending Manager 的唯一标识符。
session-id	字符串	用于标识订户会话的字符串。
source-address	字符串	从其接收消息的外部实体的地址，例如， SMSC 的 MSISDN。
streaming_content_upload_failed	字符串	此消息表明无法将流式传输内容上载到流式传输服务器。
streaming_content_delete_failed	字符串	此消息表明无法从流式传输服务器中删除流式传输内容。
subscription-expiration	日期	订阅时段结束的日期。
subscription-frequency	字符串	支付订阅费用的时间间隔。
subscription-recurring	布尔型	表示当前订阅时段结束后是否自动为下个时段向订户收费的标志。
subscription-price	浮点型	订阅时段的价格。
timestamp	时间戳	事务发生的时间。
unique-device-id	字符串	所用设备的唯一标识符。
usage-count	整型	根据为 usage-price 指定的价格而确定的使用次数。
usage-price	浮点型	根据为 usage-count 指定的使用次数而确定的支付价格。

表 2-5 事件数据 （续）

参数	数据类型	描述
user-id	字符串	启动事务的用户的唯一标识符。
username	字符串	订户的登录名。
vending-res-id	字符串	Vending Manager 用于标识内容的标记。

2.4 使用事件服务 API

要使用事件服务 API，您需要数据库客户机应用程序或 JMS 客户机应用程序。本节提供了有关根据系统需要开发相应类型客户机的信息。

2.4.1 开发数据库客户机应用程序

应用程序必须获取指向数据库的连接，以便直接查询 Oracle 数据库。具体操作细节取决于数据库服务器和客户机应用程序的本地环境。

有关连接到数据库的信息，请参见第 2-3 页上的第 2.1.1 节 “事件表”。

2.4.2 开发 JMS 客户机应用程序

要编译 JMS 客户机，请在类路径中包含 JMS 的 Java 归档 (Java Archive, JAR) 文件。此文件位于 */cds-home*/deployment/*deployment-name*/lib/jms.jar，其中 *cds-home* 是 Content Delivery Server 的安装目录， *deployment-name* 是为部署指定的名称。

要执行 JMS 客户机，请执行以下操作：

1. 在类路径中包含下表中组件的 JAR 文件。

表 2-6 执行所需的文件

JAR 文件	位置*
JMS	*/cds-home*/deployment/*deployment-name*/lib/external/javax.jms.jar
文件系统上下文 （仅当使用 Sun Java System Application Server 时需要）	*/sun-as-home*/imq/lib/fscontext.jar

* *cds-home* 是 Content Delivery Server 的安装目录。*deployment-name* 是为部署指定的名称。*sun-as-home* 是 Sun Java System Application Server 的安装目录。

2. 为 **execute** 命令指定以下选项：

- `-Dcds.home=`*cds-home*
- `-Dcds.config.file=CDS.properties`
- `-Dcds.config.dir=`*cds-home*`/deployment/`*deployment-name*`/conf`

2.5 MessageListener 样例实现

以下代码示例是 MessageListener 类的实现。此样例说明了从 CDS 记帐主题接收记帐事件时需要执行的操作。

编码样例 2-1 MessageListener 样例实现

```
package com.sun.content.server.eventservice.subscriber.internal;

import java.util.Properties;

import javax.jms.*;
import javax.naming.Context;
import javax.naming.InitialContext;

/**
 * Attach to the CDS billing topic to receive billing events
 */
public class CDSBillingSubscriber
        implements ExceptionListener, MessageListener
{
   private static final String kUSAGE = "CDSBillingSubscriber {JNDI Factory} {JNDI URL}";
    private static final String kTOPIC_CONNECTION_FACTORY_NAME =
"cds.jms.TopicConnectionFactory";
    private static final String kTOPIC_NAME = "cds.messaging.billingTopic";

    private TopicConnection                     fConnection;
    private TopicSession                        fSession;
    private boolean                             fDone = false;

    public static void main(String[] args)
    {
        if (args.length != 2)
        {
            System.out.println(kUSAGE);
            System.exit(-1);
        }
```

编码样例 2-1 MessageListener 样例实现 （续）

```
        try
        {
          CDSBillingSubscriber   billingSubscriber = new CDSBillingSubscriber();
            billingSubscriber.initJMS(args[0], args[1]);

            while (!billingSubscriber.fDone)
            {
                synchronized (billingSubscriber)
                {
                    System.out.println("Waiting...");
                    billingSubscriber.wait(1000 * 10);
                }
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
            System.exit(-1);
        }
    }


    /**
     * Initialize the JMS topic subscriber
     *
     * @param jndiFactory       the JNDI context factory
     * @param jndiProviderUrl   the JNDI connection URL
     *
     * @throws Exception
     */
    private void initJMS(String jndiFactory, String jndiProviderUrl)
        throws Exception
    {
        //  Initialize the context
        Properties  props = new Properties();
        props.put(Context.INITIAL_CONTEXT_FACTORY, jndiFactory);
        props.put(Context.PROVIDER_URL, jndiProviderUrl);
        InitialContext  context = new InitialContext(props);

        //  Get the topic connection factory and topic
        TopicConnectionFactory  topicConnectionFactory =
              (TopicConnectionFactory )context.lookup(kTOPIC_CONNECTION_FACTORY_NAME);
        Topic   topic = (Topic )context.lookup(kTOPIC_NAME);

        //  Initialize the topic connection
        fConnection = topicConnectionFactory.createTopicConnection();
        fConnection.setExceptionListener(this);
```

编码样例 2-1 MessageListener 样例实现 （续）

```
        //  Get a session and subscriber
       fSession = fConnection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
        TopicSubscriber subscriber = fSession.createSubscriber(topic);
        subscriber.setMessageListener(this);

        fConnection.start();
    }


    /**
     * Listen for messages asynchronously.Simply print them.
     *
     * @param message
     */

    public void onMessage(Message message)
    {
        try
        {
            // simply prints the message
            TextMessage txtMsg = (TextMessage )message;
            System.out.println(txtMsg.getText());
        }
        catch (JMSException e)
        {
            e.printStackTrace();
        }
    }

    /**
     * Listen for exceptions and stop waiting
     *
     * @param jmse
     */
    public void onException(JMSException jmse)
    {
        jmse.printStackTrace();
        fDone = true;
        this.notifyAll();
    }
}
```

第 3 章

记帐 API

Sun Java System Content Delivery Server 记帐 API 提供 Content Delivery Server 与记帐系统之间的接口。使用记帐 API 可以定制创建支持预付记帐或异步记帐的记帐适配器。

记帐 API 包含以下类和接口：

- `BillingManager` - 用于实现 Content Delivery Server 与记帐系统的集成的接口。`BillingManager` 接口用于处理购买授权、下载确认、订阅和取消订阅事务、下载失败后取消计费以及退还已购买内容的费用。
- `BillingInfo` - 包含记帐事务信息的类，如内容的价格和价格模型、订户标识、内容标识、开发者标识及记帐状态。此类还包括进程信息，如订户在下载某个内容项之前是否需要授权，以及在内容成功下载后是否通知记帐系统。此外，还可以使用此类访问定制字段中的信息。`BillingInfo` 对象提供记帐系统与 Content Delivery Server 之间的通信。
- `BillingException` - 记帐 API 抛出的异常。
- `BillingConstants` - 定义记帐 API 所使用的常量的类。

记帐 API 的类和接口位于 `com.sun.content.server.billing` 软件包中。有关这些类和接口的其他信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 中的 Javadoc™ 工具的 HTML 输出。

3.1 一般处理流程

记帐 API 用于处理 Content Delivery Server 与外部记帐系统之间的通信。记帐事务的详细信息保留在 `BillingInfo` 对象中。记帐适配器是 `BillingManager` 接口的实现，它用于确定事务的处理方式。

本节介绍 Content Delivery Server 与外部记帐系统之间的以下信息流：

- 内容列表
- 事务启动

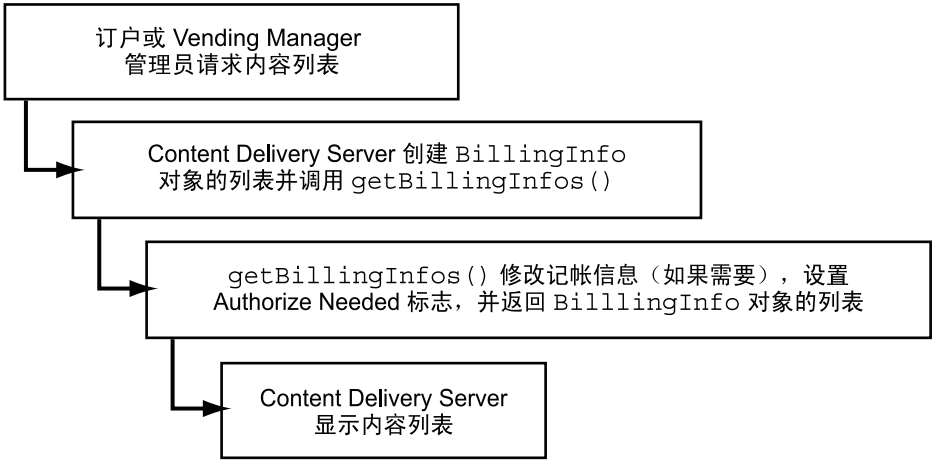
- 订户购买
- 下载确认
- 订阅验证
- 错误处理
- 定制字段访问

有关本节涉及的 `BillingManager` 方法的详细信息，请参见第 3-7 页上的第 3.2 节 “BillingManager 接口”。

3.1.1 内容列表

订户请求可用内容列表或 Vending Manager 管理员请求储存内容列表时将启动该列表进程。图 3-1 显示了此进程。

图 3-1 内容列表的进程流



以下各项提供了有关列出内容进程的其他详细信息：

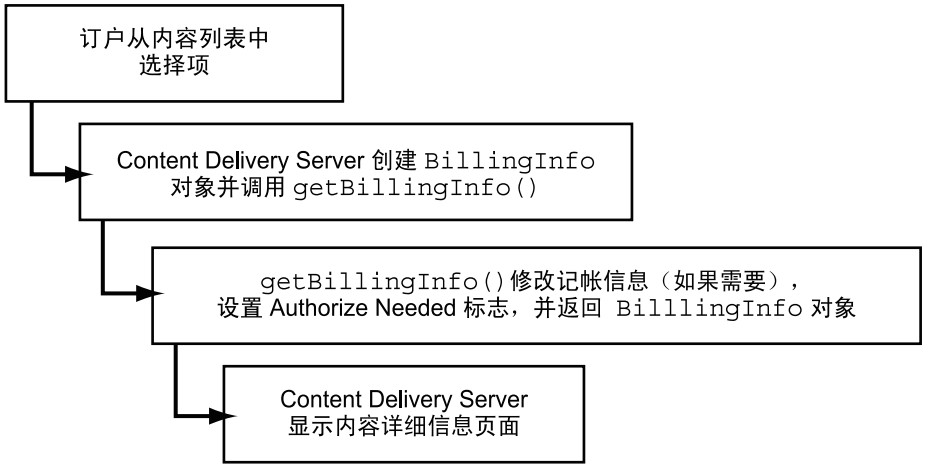
1. 当订户或 Vending Manager 管理员请求内容列表时，Content Delivery Server 将为列表中每个对象创建初始 `BillingInfo` 对象，并调用记帐适配器的 `getBillingInfos` 方法。
2. 如果需要，`getBillingInfos` 的实现可以修改购买每个事务的价格或其他详细信息。您还可以指定在购买时是否需要授权，以及在成功下载内容后是否需要确认消息。这些详细信息在返回到 Content Delivery Server 的 `BillingInfo` 对象中进行设置。有关其他信息，请参见第 3-9 页上的第 3.2.6 节 “getBillingInfos 方法”。

3. Content Delivery Server 使用每个返回的 `BillingInfo` 对象展示所显示的列表中的定价信息。

3.1.2 事务启动

当订户单击内容项旁边的“查看详细信息”时，将会启动记帐事务。图 3-2 显示了此进程。

图 3-2 事务启动的进程流



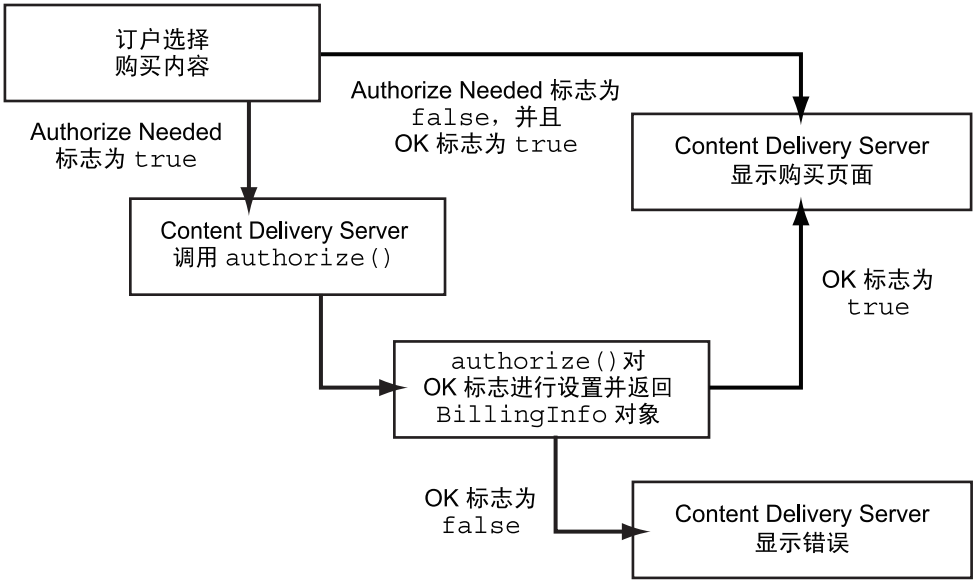
以下各项提供了有关事务启动进程的其他详细信息：

1. 当订户请求项目的详细信息时， Content Delivery Server 将根据 Content Delivery Server 数据库中的信息为订户的选择创建初始 `BillingInfo` 对象。
2. 然后 Content Delivery Server 将调用记帐适配器的 `getBillingInfo` 方法，并将 `BillingInfo` 对象传递给该方法。
3. 如果需要， `getBillingInfo` 方法的实现可以修改购买价格或事务的其他详细信息。您还可以指定在购买时是否需要授权，以及在成功下载内容后是否需要确认消息。这些详细信息在返回到 Content Delivery Server 的 `BillingInfo` 对象中进行设置。有关其他信息，请参见第 3-9 页上的第 3.2.5 节 “getBillingInfo 方法”。
4. Content Delivery Server 使用返回的 `BillingInfo` 对象向订户展示选定项目的内容详细资料和定价信息。

3.1.3 订户购买

当订户选择某个内容项并单击 “购买”时，将会启动购买进程。购买进程将使用启动事务时创建的 `BillingInfo` 对象。图 3-3 显示了此进程。

图 3-3 订户购买的进程流



以下各项提供了有关订户购买进程的其他详细信息：

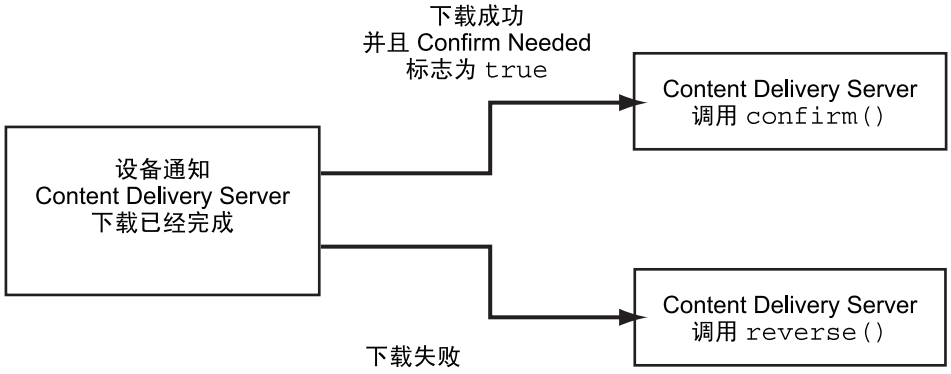
1. 如果订户选择购买内容， Content Delivery Server 将检查由 `getBillingInfo` 方法返回的 `BillingInfo` 对象，以查看是否需要授权。
2. 如果需要授权， Content Delivery Server 将调用记帐适配器的 `authorize` 方法，并传递 `BillingInfo` 对象。此方法用于确定是否允许订户购买内容，并在 `BillingInfo` 对象中设置 "OK" 标志。然后该对象将返回到 Content Delivery Server。有关其他信息，请参见第 3-7 页上的第 3.2.1 节 “authorize 方法”。
3. 如果订户被授权购买内容或无需授权，则显示购买页面。如果订户未被授权购买内容，则发出错误消息。

订户购买内容时，与事务关联的 `BillingInfo` 对象存储在 Content Delivery Server 数据库中。如有任何进一步操作需要记帐信息，Content Delivery Server 就会以订户 ID 和内容 ID 为关键字从数据库中检索对象。

3.1.4 下载确认

当设备完成下载并通知 Content Delivery Server 时，将会启动确认进程。图 3-4 显示了此进程。如果下载成功并且记帐系统不需要通知，则不执行任何操作。

图 3-4 下载确认的进程流



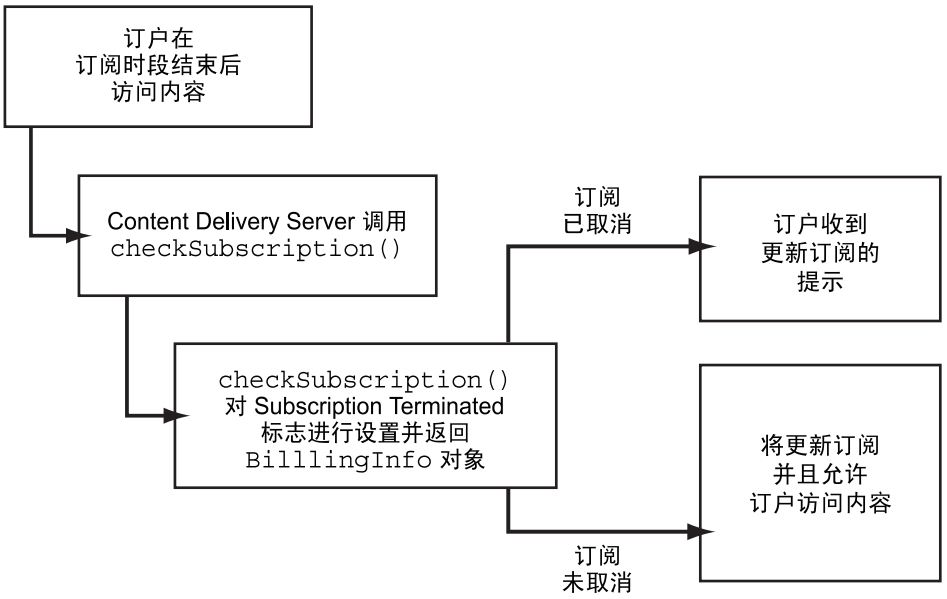
以下各项提供了有关下载确认进程的其他详细信息：

1. 当 Content Delivery Server 从设备收到下载成功的确认消息时， Content Delivery Server 将检查记帐信息，以查看是否需要将该确认消息通知记帐系统。
2. 如果需要确认，Content Delivery Server 将调用记帐适配器的 `confirm` 方法。有关其他信息，请参见第 3-8 页上的第 3.2.3 节 “confirm 方法”。
3. 如果设备返回的不是成功确认的消息，而是错误消息， Content Delivery Server 将调用记帐适配器的 `reverse` 方法，并传递此事务的 `BillingInfo` 对象。对 `reverse()` 实现进行编码，以确保订户不会为未下载的内容付费。有关其他信息，请参见第 3-10 页上的第 3.2.8 节 “reverse 方法”。

3.1.5 订阅验证

当订户在订阅时段结束后访问内容时，将会启动订阅验证进程。图 3-5 显示了此进程。

图 3-5 订阅验证的进程流



以下各项提供了有关订阅验证进程的其他详细信息：

1. 当订户在订阅时段结束后访问内容时， Content Delivery Server 将调用 `checkSubscription` 方法，以查看是否在 Content Delivery Server 外部取消了订阅。
2. 对 `checkSubscription` 方法实现进行编码，以便在返回到 Content Delivery Server 的 `BillingInfo` 对象中设置 "Subscription Terminated" 标志。有关其他信息，请参见第 3-8 页上的第 3.2.2 节 “checkSubscription 方法”。
3. 如果 `BillingInfo` 对象中的 "Subscription Terminated" 标志为 `true`，或 Content Delivery Server 的订阅状态为 `canceled`，订户将收到更新订阅的提示。如果该标志为 `false` 且状态不是 `canceled`，订阅将自动更新并运行该内容。

3.1.6 错误处理

当记帐系统检测到错误时，将向订户显示普通错误消息。通过在由 `BillingManager` 接口的实现返回的 `BillingInfo` 对象中设置回复消息，可以定制此消息以显示附加信息。只要记帐适配器检测到错误，即调用 `setOK` 方法将 `BillingInfo` 对象中的 "OK" 标志设置为 false，并调用 `setReplyMessage` 方法以设置向订户显示的消息。

3.1.7 定制字段访问

有关内容的信息存储在 Content Delivery Server 提供的数据字段中，例如，名称、说明和内容类型。可以将实现所需的其他信息存储在定义的定制字段中。可以使用 `BillingInfo` 对象访问有关内容的信息，包括定制字段中的信息。

当 Content Delivery Server 创建 `BillingInfo` 对象时，将会调用 `setReplyOthers` 方法，以便将定制字段添加到散列表中，该表包含特定于实现的信息。添加到表中的每个字段的键与用于定义定制字段的键相同。要从记帐管理器中访问这些字段，请调用 `getReplyOthers` 方法，然后使用定制字段键提取散列表中的项目。

有关设置定制字段的信息，请参见《Sun Java™ System Content Delivery Server 5.1 集成和配置指南》中的第 1.8 节。

3.2 BillingManager 接口

`BillingManager` 接口根据 `BillingInfo` 对象中包含的事务详细信息处理记帐事务。实现 `BillingManager` 以创建系统的记帐适配器。此记帐适配器是 Content Delivery Server 与记帐系统之间的接口。

3.2.1 authorize 方法

```
abstract BillingInfo authorize(BillingInfo inBillingInfo, boolean[]
inNeedToAuthorizeBillingModel)
```

如果外部记帐系统需要对事务进行授权，则当订户单击项目旁边的 “购买”按钮时，Content Delivery Server 将调用 `authorize` 方法。要表明需要授权，必须通过 `getBillingInfo` 方法设置 `BillingInfo` 对象中的 "Authorize Needed" 标志。

使用此方法可以确定订户是否已被授权购买所请求的内容。如果记帐系统支持预付记帐模型，则使用此方法可以检验订户帐户中是否有足够的余额购买该内容。

参数 `inNeedToAuthorizeBillingModel` 提供一个标志数组，这些标志按 `BillingConstants` 类中定义的以下价格模型常量进行了编索：

- `DOWNLOAD` - 按下载收费
- `SUBSCRIPTION` - 按订阅收费
- `TRIAL` - 提供免费试用
- `USAGE` - 按使用次数收费
- `LIMITED_TIME` - 按特定时间间隔收费

每个标志表示在授权事务时是否考虑价格模型。例如，假设已给定以下参数：

- 价格模型为 “下载”
- 仅在第一次下载时收取订户的费用
- 当前记帐事务是订户的第二次下载

将按 DOWNLOAD 编索的标志设置为 `false`，以表明此事务不收取下载费用。因此，在授权该事务时无需考虑下载费用。根据系统的需要使用或忽略这些标志。

如果订户被授权购买内容，则可以通过在将 `BillingInfo` 对象返回到 Content Delivery Server 之前调用 `setIsOK` 方法，将 `BillingInfo` 对象中的 "OK" 标志设置为 `true`。如果未授权订户购买该内容，则将 "OK" 标志设置为 `false`。

如果未通过 `getBillingInfo` 方法设置 `BillingInfo` 对象中的 "Confirm Needed" 标志，则可以通过调用 `setConfirmNeeded` 方法来设置该标志。要在将内容成功下载到设备时通知记帐系统，请将该标志设置为 `true`。如果不希望通知记帐系统，请将该标志设置为 `false`。

3.2.2 checkSubscription 方法

```
abstract BillingInfo checkSubscription(BillingInfo inBillingInfo)
throws BillingException
```

如果订户试图在订阅时段结束后使用内容， Content Delivery Server 将调用 `checkSubscription` 方法。使用此方法可以通知 Content Delivery Server 已在 Content Delivery Server 外部终止订阅。

对实现进行编码，以便通过调用 `BillingInfo` 对象的 `setSubscriptionTerminated` 方法来设置 "Subscription Terminated" 标志。要表明已终止订阅，请将该标志设置为 `true`。然后订户将收到更新订阅的提示。要表明订阅仍然有效，请将该标志设置为 `false`。然后在下一期间将自动更新此订阅。

3.2.3 confirm 方法

```
abstract BillingInfo confirm(BillingInfo inBillingInfo)
```

如果外部记帐系统要求收到下载成功的通知， Content Delivery Server 收到订户设备的下载确认后将调用 `confirm` 方法。要表明需要确认，必须通过 `getBillingInfo` 方法或 `authorize` 方法设置 `BillingInfo` 对象中的 "Confirm Needed" 标志。

确认内容已下载后，可以使用此方法执行所需的操作。例如，您可能希望仅在收到下载成功的确认后从订户帐户扣除金额。

3.2.4 contentDelete 方法

```
public abstract void contentDelete(BillingInfo inBillingInfo) throws
BillingException
```

从设备中删除内容后，Content Delivery Server 将调用 `contentDelete` 方法。实现此方法可通知记帐系统，订户不再使用该内容。

3.2.5 getBillingInfo 方法

```
abstract BillingInfo getBillingInfo(BillingInfo inBillingInfo)
```

当订户请求项目的详细信息或购买项目时， Content Delivery Server 将调用 `getBillingInfo` 方法。 Content Delivery Server 创建 `BillingInfo` 对象并将其传递给此方法，该对象包括记帐信息。

可以使用此方法根据需要修改记帐信息，并将修改后的 `BillingInfo` 对象返回到 Content Delivery Server。例如，要向选定订户提供折扣，请更改 Content Delivery Server 指定的价格。

对实现进行编码，以便通过调用 `BillingInfo` 对象的 `setAuthorizeNeeded` 方法来设置 "Authorize Needed" 标志。要让记帐系统验证订户是否已被授权购买选定内容，请将该标志设置为 `true`。如不希望对订户进行预授权，请将该标志设置为 `false`。

如果将 "Authorize Needed" 标志设置为 `false`，请在此方法中设置 "Confirm Needed" 标志和 "OK" 标志。要在将内容成功下载到设备时通知记帐系统，请通过调用 `setConfirmNeeded` 方法将 "Confirm Needed" 标志设置为 `true`。如果不希望通知记帐系统，请将该标志设置为 `false`。

要使订户能够在 "Authorize Needed" 标志为 `false` 时购买内容，请通过调用 `setOK` 方法将 "OK" 标志设置为 `true`。如果将 "OK" 标志设置为 `false` 并且 "Authorize Needed" 标志也是 `false`，则不允许订户下载选定内容。如果将 "Authorize Needed" 标志设置为 `true`，请在 `authorize` 方法中设置 "OK" 标志。

3.2.6 getBillingInfos 方法

```
abstract BillingInfo[] getBillingInfos(BillingInfo[] inBillingInfos)
```

当订户请求可用内容列表或 Vending Manager 管理员请求储存内容列表时， Content Delivery Server 将调用 `getBillingInfos` 方法。 Content Delivery Server 创建 `BillingInfo` 对象列表并将其传递给此方法，该对象包括记帐信息。

使用此方法可以根据需要修改记帐信息，并将修改过的 `BillingInfo` 对象列表返回到 Content Delivery Server。例如，要提供折扣，请更改 Content Delivery Server 指定的价格。

对实现进行编码，以便通过调用列表中每个 BillingInfo 对象的 setAuthorizeNeeded 方法来设置 "Authorize Needed" 标志。要让记帐系统验证订户是否已被授权购买选定内容，请将该标志设置为 true。如不希望对订户进行预授权，请将该标志设置为 false。

如果将 "Authorize Needed" 标志设置为 false，请在此方法中设置 "Confirm Needed" 标志和 "OK" 标志。要在将内容成功下载到设备时通知记帐系统，请通过调用 setConfirmNeeded 方法将 "Confirm Needed" 标志设置为 true。如果不希望通知记帐系统，请将该标志设置为 false。

要使订户能够在 "Authorize Needed" 标志为 false 时购买内容，请通过调用 setOK 方法将 "OK" 标志设置为 true。如果将 "OK" 标志设置为 false 并且 "Authorize Needed" 标志也是 false，则不允许订户下载选定内容。如果将 "Authorize Needed" 标志设置为 true，请在 authorize 方法中设置 "OK" 标志。

3.2.7 refund 方法

```
abstract void refund(BillingInfo inBillingInfo)
```

当客户服务代理使用 Vending Manager 管理控制台退还订户的购买费用时， Content Delivery Server 将调用 refund 方法。Content Delivery Server 通过存储在数据库中的记帐信息为原始记帐事务创建 BillingInfo 对象，并将其传递给此方法。

使用此方法可以执行贷记订户帐户所需的操作。

3.2.8 reverse 方法

```
abstract void reverse(BillingInfo inBillingInfo)
```

当无法将内容下载到订户设备时， Content Delivery Server 将调用 reverse 方法。Content Delivery Server 通过存储在数据库中的记帐信息为原始记帐事务创建 BillingInfo 对象，并将其传递给此方法。

使用此方法可以取消记帐事务，因此订户无需为该内容付费。

3.2.9 subscribe 方法

```
abstract void subscribe(BillingInfo inBillingInfo)
```

当订户开始订阅内容时， Content Delivery Server 将调用 subscribe 方法。 Content Delivery Server 通过存储在数据库中的记帐信息为原始记帐事务创建 BillingInfo 对象，并将其传递给此方法。

使用此方法可以为订户启动订阅。如果订阅重复出现，则此方法仅在订阅启动时调用一次。在每次订阅时段结束时，记帐系统必须记得向订户收取费用。如果订阅不重复出现，则在每次订户更新订阅时调用该方法。

3.2.10 unsubscribe 方法

```
abstract void unsubscribe(BillingInfo inBillingInfo)
```

当订户取消重复出现的订阅时， Content Delivery Server 将调用 `unsubscribe` 方法。Content Delivery Server 通过存储在数据库中的记帐信息为原始记帐事务创建 `BillingInfo` 对象，并将其传递给此方法。

使用此方法可以在订阅时段结束时停止自动付费。

3.3 使用记帐 API

`cdsapi.jar` 文件中提供了记帐 API 的类。在编译适配器时，此文件必须位于类路径中。为方便起见， `$CDS_HOME/dist/cds/staging/jar` 目录中提供了所有 Content Delivery Server JAR 文件的副本。在编译所创建的适配器时，请在类路径中使用此缓冲区。

使适配器可用于 Content Delivery Server 取决于所使用的应用服务器，以及是否已部署了该服务器。要使适配器可用，请执行以下操作：

1. **为适配器创建 JAR 文件。**
2. **对于所有应用服务器，请将该 JAR 文件放在 `$CDS_HOME/dist/cds/lib/external` 目录中。**

   现在，该适配器将包含在所有以后的部署中。
3. **如果现有部署需要使用该适配器，请将该 JAR 文件放在每个部署的 `$CDS_HOME/deployment/`*`deployment-name`*`/lib/external` 目录中。**

   如果使用的是 WebLogic Server，则会为您处理类路径。

   如果使用的是 Sun Java System Application Server，则会为每个部署更新类路径：

   a. **在编辑 `$CDS_HOME/deployment/`*`deployment-name`*`/sun/domains/cdsdomain/config/domain.xml` 文件之前，先对其进行备份，以便从编辑期间可能引入的任何错误中恢复。**

   b. **编辑 `domain.xml` 并修改 `java-config` 元素，将 JAR 文件的绝对路径添加到 `classpath-suffix` 属性中。**

   c. **保存所做的更改。**

4. **编辑** `$CDS_HOME/deployment/`*deployment-name*`/conf` **目录中的** `security.config` **文件：**

   a. **将** `module.security.billingmanager` **属性设置为** `BillingManager` **类实现的类名，例如：**

   ```
   module.security.billingmanager=
   com.sun.content.server.billing.external.MyBillingManager
   ```

   b. **保存所做的更改。**

5. **重新启动任何现有部署以识别新的 JAR 文件。**

3.4 样例记帐适配器

编码样例 3-1 显示了 `BillingManager` 的默认实现。

编码样例 3-1 样例 `BillingManager` 实现

```
package com.sun.content.server.billing.external;

import com.sun.content.server.billing.BillingException;
import com.sun.content.server.billing.BillingInfo;
import com.sun.content.server.billing.BillingManager;

/**
 * This is a sample implementation of the Billing API.
 */
public class CDSBillingManager implements BillingManager
{
  // These flags can be used to simulate responses from the billing
  //integration.
  public static final int SUCCESS = 0;
  public static final int EXCEPTION = 1;
  public static final int BILLING_EXCEPTION = 2;
  public static final int UNAUTHORIZED = 3;
  public static final int NULL = 4;

  // by default everything will pass through fine
  // but you can change these at runtime.
  public static int AUTHORIZE_RESPONSE = SUCCESS;
  public static int GET_BILLING_INFO_RESPONSE = SUCCESS;
  public static int GET_BILLING_INFOS_RESPONSE = SUCCESS;
  public static int CONFIRM_RESPONSE = SUCCESS;
  public static int DELETE_RESPONSE = SUCCESS;
  public static int REFUND_RESPONSE = SUCCESS;
```

编码样例 3-1 样例 BillingManager 实现 （续）

```
  public static int REVERSE_RESPONSE = SUCCESS;
  public static int SUBSCRIBE_RESPONSE = SUCCESS;
  public static int UNSUBSCRIBE_RESPONSE = SUCCESS;
  public static int CHECK_SUBSCRIPTION_RESPONSE = SUCCESS;

  /**
   * see BillingManager#getBillingInfo(BillingInfo)
   */
  public BillingInfo getBillingInfo(BillingInfo inBillingInfo)
         throws BillingException
  {
    if (GET_BILLING_INFO_RESPONSE == NULL)
      return null;

    if (GET_BILLING_INFO_RESPONSE == EXCEPTION)
      throw new NullPointerException("Developer Null Pointer");

    if (GET_BILLING_INFO_RESPONSE == BILLING_EXCEPTION)
       throw new BillingException("Developer Billing Exception");

    // Set IsAuthorizeNeeded flag
    inBillingInfo.setAuthorizeNeeded(true);
    return inBillingInfo;
  }

  /**
   * see BillingManager#getBillingInfos(BillingInfo[])
   */
  public BillingInfo[]
    getBillingInfos(BillingInfo[] inBillingInfos)
    throws BillingException
  {
      for (int index = 0; index < inBillingInfos.length; index++)
    {
      // Set IsAuthorizeNeeded flag
      inBillingInfos[index].setAuthorizeNeeded(true);
    }

    if (GET_BILLING_INFOS_RESPONSE == NULL)
      return null;

    if (GET_BILLING_INFOS_RESPONSE == EXCEPTION)
      throw new NullPointerException("Testing Null Pointer");

    if (GET_BILLING_INFOS_RESPONSE == BILLING_EXCEPTION)
       throw new BillingException("Testing Billing Exception");
```

编码样例 3-1 样例 BillingManager 实现 （续）

```
    return inBillingInfos;
  }

  /**
   * see.BillingManager#authorize(BillingInfo, boolean[])
   */
  public BillingInfo authorize(BillingInfo inBillingInfo,
         boolean[] inNeedToAuthorizeBillingModel)
         throws BillingException
  {
    if (AUTHORIZE_RESPONSE == NULL)
      return null;

    if (AUTHORIZE_RESPONSE == EXCEPTION)
      throw new NullPointerException("Testing Null Pointer");

    if (AUTHORIZE_RESPONSE == BILLING_EXCEPTION)
     throw new BillingException("Testing Billing Exception");

    if (AUTHORIZE_RESPONSE == UNAUTHORIZED)
    {
      inBillingInfo.setOk(false);
      inBillingInfo.setReplyMessage("You are not authorized");
      return inBillingInfo;
    }

// Set IsOk and IsConfirmNeeded flags
    inBillingInfo.setConfirmNeeded(true);
    inBillingInfo.setOk(true);

    return inBillingInfo;
  }

  /**
   * seeBillingManager#confirm(BillingInfo)
   */
  public BillingInfo confirm(BillingInfo inBillingInfo)
      throws BillingException
  {
    if (CONFIRM_RESPONSE == EXCEPTION)
      throw new NullPointerException("Developer Null Pointer");

    if (CONFIRM_RESPONSE == BILLING_EXCEPTION)
       throw new BillingException("Developer Billing Exception");
    return inBillingInfo;
  }
```

编码样例 3-1 样例 BillingManager 实现 （续）

```
  /**
   * see BillingManager#reverse(BillingInfo)
   */
  public void reverse(BillingInfo inBillingInfo)
      throws BillingException
  {
    if (REVERSE_RESPONSE == EXCEPTION)
      throw new NullPointerException("Developer Null Pointer");

    if (REVERSE_RESPONSE == BILLING_EXCEPTION)
       throw new BillingException("Developer Billing Exception");
  }

  /**
   * see BillingManager#refund(BillingInfo)
   */
  public void refund(BillingInfo inBillingInfo)
      throws BillingException
  {
    if (REFUND_RESPONSE == EXCEPTION)
      throw new NullPointerException("Developer Null Pointer");

    if (REFUND_RESPONSE == BILLING_EXCEPTION)
       throw new BillingException("Developer Billing Exception");
  }

  /**
   * seeBillingManager#subscribe(BillingInfo)
   */
  public void subscribe(BillingInfo inBillingInfo)
      throws BillingException
  {
    if (SUBSCRIBE_RESPONSE == EXCEPTION)
      throw new NullPointerException("Developer Null Pointer");

    if (SUBSCRIBE_RESPONSE == BILLING_EXCEPTION)
       throw new BillingException("Developer Billing Exception");
  }

  /**
   * see BillingManager#unsubscribe(BillingInfo)
   */
  public void unsubscribe(BillingInfo inBillingInfo)
      throws BillingException
  {
    if (UNSUBSCRIBE_RESPONSE == EXCEPTION)
      throw new NullPointerException("Developer Null Pointer");
```

编码样例 3-1 样例 BillingManager 实现 （续）

```
    if (UNSUBSCRIBE_RESPONSE == BILLING_EXCEPTION)
       throw new BillingException("Developer Billing Exception");
  }

  /**
   * see BillingManager#checkSubscription(BillingInfo)
   */
  public BillingInfo checkSubscription(BillingInfo inBillingInfo)
      throws BillingException
  {
    if (CHECK_SUBSCRIPTION_RESPONSE == NULL)
      return null;

    if (CHECK_SUBSCRIPTION_RESPONSE == EXCEPTION)
      throw new NullPointerException("Developer Null Pointer");

    if (CHECK_SUBSCRIPTION_RESPONSE == BILLING_EXCEPTION)
       throw new BillingException("Developer Billing Exception");

    inBillingInfo.setSubscriptionTerminated(false);

    return inBillingInfo;
  }

  /**
   * see BillingManager#contentDelete(BillingInfo)
   */
  public void contentDelete(BillingInfo inBillingInfo)
      throws BillingException
  {
    if (DELETE_RESPONSE == EXCEPTION)
     throw new NullPointerException("Developer Null Pointer");

    if (DELETE_RESPONSE == BILLING_EXCEPTION)
     throw new BillingException("Developer Billing Exception");
  }
}
```

第 4 章

内容管理 API

Sun Java System Content Delivery Server 内容管理 API 提供 Content Delivery Server 与内容管理系统之间的接口。使用此 API 可以编写内容管理适配器，以便在内容发送到订户的设备时测试内容二进制代码或更改内容信息。例如，可以创建内容管理适配器，以便添加用于处理数字权限管理 (Digital Rights Management, DRM) 的代码，或者在预览文件中添加水印。

注 – 如果内容管理适配器需要在开发者提交内容的原始版本上运行，则提交验证器工作流不得在提交时执行任何测试或修改。有关创建工作流和配置 Content Delivery Server 附带的提交验证器工作流的信息，请参见《Sun Java™ System Content Delivery Server 5.1 集成和配置指南》中的第 14 章 “提交验证器工作流”。

内容管理适配器在处理收到的调用时需要一些时间，这会延迟内容传送到订户的时间。尽量将内容管理 API 的使用限制为不增加过多开销的操作。

内容管理 API 包括以下元素：

- `ContentManager` - 用于实现 Content Delivery Server 与内容管理系统的集成的接口。 `ContentManager` 接口提供了多种方法以访问有关内容的信息。
- `ContentInfo` - 此类包含内容对象的信息，如内容描述符和二进制代码、内容描述符和二进制代码的 MIME 类型、内容的大小和类型（如 MIDlet、ringtone 等）以及事务详细信息，如一步或两步下载。
- `ContentPreview` - 此类包含预览文件的信息，例如，文件的 MIME 类型、二进制文件、上次修改日期以及其他属性。
- `MimeType` - 此类包含 MIME 类型的信息，例如，文件扩展名和 MIME 类型字符串。
- `ContentException` - 内容管理 API 抛出的异常。使用此类可以报告在处理内容信息时或是在购买事务中出现的错误。
- `ContentConstants` - 此类定义了 `ContentInfo` 对象用于访问内容元数据的键值。

有关这些类和接口的其他信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 中的 Javadoc 工具的 HTML 输出。

4.1 一般处理流程

内容管理 API 用于处理 Content Delivery Server 与外部内容管理系统或 DRM 服务器之间的通信。内容和事务的详细信息保留在 `ContentInfo` 对象中。使用此 API 可以访问和处理内容传送给订户时的内容信息。

本节介绍了设备、 Content Delivery Server 及内容管理适配器之间的信息流。将讨论以下主题：

- 获取内容列表
- 获取内容详细资料
- 下载内容

4.1.1 获取内容列表

当订户在 Subscriber Portal 中单击 “类别”下的内容标题时，将启动内容列表请求。为了实现该请求，系统将执行以下操作：

1. 订户设备向 Content Delivery Server 发送请求以获取有关选定类别中内容项的信息。
2. Content Delivery Server 根据 Content Delivery Server 数据库中的信息，为列表中的各项创建初始 `ContentInfo` 对象。此信息显示给订户或由 Content Delivery Server 使用。
3. Content Delivery Server 调用 `getContentInfos` 并传递列表中各项的 `ContentInfo` 对象。

   `getContentInfos` 的实现可以根据系统需要更改内容信息。此时仅修改内容列表中显示给订户的信息。
4. Content Delivery Server 将该类别的项目列表和其他信息返回到订户设备。
5. Subscriber Portal 在页面的 “结果”部分中显示该列表。

4.1.2 获取内容详细资料

当订户在 Subscriber Portal 的 “结果”页中单击某个内容项的名称或 “更多详细信息”时，将启动有关特定内容项详细信息的请求。以下各项介绍了有关获取内容详细资料的事务步骤：

1. 订户在 Subscriber Portal 的“结果”页中单击特定内容项的名称或“更多详细信息”。
2. 订户设备向 Content Delivery Server 发送请求以获取选定内容项的信息。

3. Content Delivery Server 为单个项目调用 `getContentInfo`。方法的实现可以返回有关内容的详细信息，如二进制代码和描述符的 MIME 类型、内容的大小和类型（如 ringtone 或 MIDlet）以及事务详细信息。也可以修改与内容项关联的任何预览文件。

4. Content Delivery Server 将有关内容项以及任何预览文件的详细信息传递到订户设备。

4.1.3 下载内容

订户从设备选择购买内容时将启动获取内容的进程。通常，在将内容传送给订户之前将对其进行测试。可以基于很多因素进行测试，如订户的电话类型 （CDMA 或 GSM）、价格模型 （订阅或使用）、内容类型或其他特征。以下各项介绍了下载内容的事务步骤：

1. 订户在 Subscriber Portal 中选择所需的内容项，然后单击 “立即下载”。订户必须从设备而不是 PC 访问 Subscriber Portal。

2. 对于两步下载，设备会将请求发送到 Content Delivery Server 以下载内容描述符。

3. Content Delivery Server 为请求的内容项生成初始的 `ContentInfo` 对象。

4. Content Delivery Server 将从 Content Delivery Server 数据库中检索内容二进制代码、描述符及其 MIME 类型，并将其传递到内容管理系统。

5. Content Delivery Server 将调用`getContentDescriptor` 以处理内容详细资料并对描述符文件进行适当更新。

6. 内容管理系统将返回描述符文件的更新版本，该更新版本包含向订户传送的值。

7. Content Delivery Server 将更新的描述符文件传递到订户设备。

8. 该设备向 Content Delivery Server 发出请求，以获得内容二进制代码。订户在设备中单击显示的 URL 即可发送请求 （发送请求的方法之一）。

9. Content Delivery Server 使用内容二进制代码及其 MIME 类型填充 `ContentInfo` 对象并将该对象传递给内容管理适配器。

10. Content Delivery Server 调用 `getContentBinary` 以创建更新的内容二进制文件。

11. 内容管理适配器对内容进行测试并将更新的内容二进制文件及二进制 MIME 类型传回 Content Delivery Server。

注 – 在发布内容时将完成功能匹配 （用于确定运行内容所用的设备）。内容管理适配器更改内容后不能出现以下情形，即设备无法再运行该内容。可能影响设备运行内容的更改包括增加内容大小或更改 MIME 类型。

---

12. Content Delivery Server 将更新内容传递到订户设备。

4.2 ContentManager 接口

ContentManager 接口提供 Content Delivery Server 与内容管理系统或 DRM 服务器之间的接口。您可以实现它所提供的方法，以便在内容二进制代码、描述符和其他信息传送给订户之前对其进行修改。

该类中的方法将 ContentInfo 对象和 BillingInfo 对象作为参数。 ContentInfo 对象包含内容二进制代码、内容描述符文件和其他信息，如内容二进制代码和描述符的 MIME 类型、内容的大小和类型，以及下载内容所需的步骤数。

BillingInfo 对象包含事务详细信息，如价格模型。此对象还包含订户信息。有关记帐和 BillingInfo 对象的详细信息，请参见第 3 章。

ContentManager 接口位于 com.sun.content.server.content 软件包中。

4.2.1 getContentInfo 方法

```
abstract ContentInfo getContentInfo(ContentInfo inContentInfo,
BillingInfo inBillingInfo) throws ContentException
```

此方法在订户请求有关单个内容项的信息时由 Content Delivery Server 调用。ContentInfo 对象包含有关初始内容项的信息，如内容二进制代码和描述符的 MIME 类型、估计的内容大小和类型（如 ringtone 或 MIDlet）以及下载该内容所需的步骤数。

要修改预览文件，请使用 ContentInfo.getValue 方法和 KEY_CONTENT_PREVIEWS 键获取 ContentInfo 对象中的原始 ContentPreview 对象集合。每个 ContentPreview 对象表示一个预览文件。如果内容项没有预览文件，该方法将返回一个空集合。在从 Subscriber Portal 请求内容详细信息时，如果既未包含返回资源二进制文件的标志，也未包含返回资源 URL 的标志，该方法将返回 Null。

在修改预览文件后，请使用 ContentInfo.setValue 方法和 KEY_CONTENT_PREVIEWS 键保存返回到 Content Delivery Server 的 ContentInfo 对象中的已修改 ContentPreview 对象集合。

4.2.2 getContentInfos 方法

```
abstract ContentInfo[] getContentInfos(ContentInfo[]inContentInfos,
BillingInfo[] inBillingInfos) throws ContentException
```

此方法在订户请求有关类别中内容项列表的信息时由 Content Delivery Server 调用。对实现进行编码，以便仅修改内容列表中显示的信息。

4.2.3 getContentDescriptor 方法

```
abstract ContentInfo getContentDescriptor(ContentInfo inContentInfo,
BillingInfo inBillingInfo) throws ContentException
```

此方法在订户启动将内容描述符下载到设备的请求时由 Content Delivery Server 调用。Content Delivery Server 检索内容描述符文件。内容描述符和二进制代码将传递到内容管理系统。内容管理系统可以更新内容描述符信息（包括大小），并将其返回到 Content Delivery Server 以便传送给订户。

4.2.4 getContentBinary 方法

```
abstract ContentInfo getContentBinary(ContentInfo inContentInfo,
BillingInfo inBillingInfo) throws ContentException
```

此方法在订户启动将内容二进制代码下载到设备的请求时由 Content Delivery Server 调用。在调用 `getContentBinary` 时可以将二进制代码传递到内容管理适配器。要传送给订户的已测试内容二进制代码必须返回到 Content Delivery Server。

4.3 使用内容管理 API

`cdsapi.jar` 中提供了内容管理 API 的类。编译内容管理适配器时， `cdsapi.jar` 文件必须位于类路径中。为方便起见， `$CDS_HOME/dist/cds/staging/jar` 目录中提供了所有 Content Delivery Server JAR 文件的副本。在编译所创建的适配器时，请在类路径中使用此缓冲区。

使适配器可用于 Content Delivery Server 取决于所使用的应用服务器，以及是否已部署了该服务器。要使适配器可用，请执行以下操作：

1. **为适配器创建 JAR 文件。**
2. **对于所有应用服务器，请将该 JAR 文件放在 `$CDS_HOME/dist/cds/lib/external` 目录中。**

   现在，该适配器将包含在所有以后的部署中。
3. **如果现有部署需要使用该适配器，请将该 JAR 文件放在每个部署的 `$CDS_HOME/deployment/`*deployment-name*`/lib/external` 目录中。**

   如果使用的是 WebLogic Server，则会为您处理类路径。

   如果使用的是 Sun Java System Application Server，则会为每个部署更新类路径：

a. **在编辑 `$CDS_HOME/deployment/`*deployment-name*`/sun/domains/cdsdomain/config/domain.xml` 文件之前，先对其进行备份，以便从编辑期间可能引入的任何错误中恢复。**

b. **编辑 `domain.xml` 并修改 `java-config` 元素，将 JAR 文件的绝对路径添加到 `classpath-suffix` 属性中。**

c. **保存所做的更改。**

4. **编辑 `$CDS_HOME/deployment/`*deployment-name*`/conf` 目录中的 `security.config` 文件：**

a. **将名为 `module.security.contentmanager` 的属性添加到具有相同名称的现有属性之后。**

现有 `module.security.contentmanager` 属性指向 Content Delivery Server 附带的 DRM 代理所使用的具体实现，该属性必须第一个出现。

b. **将所添加的属性设置为 `ContentManager` 接口实现的全限定软件包和类名。**

以下代码显示了 `module.security.contentmanager` 属性的样例设置：

```
module.security.contentmanager=
com.sun.content.server.fulfillment.content.external.SunContentManager
module.security.contentmanager=myapps.adapters.ContentManagerImpl
```

c. **将 `module.security.contentmanager.enabled` 属性设置为 `true`，以表明适配器可供 Content Delivery Server 调用。**

d. **保存所做的更改。**

5. **重新启动任何现有部署以识别新的 JAR 文件。**

4.4 样例内容管理适配器

编码样例 4-1 显示了 `ContentManager` 接口的样例实现。

编码样例 4-1 样例 `ContentManager` 实现

```
import com.sun.content.server.content.*;
import com.sun.content.server.billing.BillingInfo;

public class ContentManagerImpl implements ContentManager
{
  public ContentInfo getContentInfo(
    ContentInfo inContentInfo,
```

编码样例 4-1 样例 ContentManager 实现 （续）

```
    BillingInfo inBillingInfo)
  throws ContentException
  {
    // Update the information that is shown to the user
    return inContentInfo;
  }

  public ContentInfo[] getContentInfos(
    ContentInfo[] inContentInfos,
    BillingInfo[] inBillingInfos)
  throws ContentException
  {
    // Iterate through each ContentInfo object and update the
    // information that is shown to the user when a list of
    // content is shown.
    return inContentInfos;
  }

  public ContentInfo getContentDescriptor(
    ContentInfo inContentInfo,
    BillingInfo inBillingInfo)
  throws ContentException
  {
    // Update content download descriptor
    return inContentInfo;
  }

  public ContentInfo getContentBinary(
    ContentInfo inContentInfo,
    BillingInfo inBillingInfo)
  throws ContentException
  {
    // Update content binary, binary MIME type
    return inContentInfo;
  }
}
```

第 5 章

内容验证 API

本章介绍 Sun Java System Content Delivery Server 内容验证 API。使用此 API 可创建验证和保护所提交内容时所需的内容验证适配器。在将内容提交到 Content Delivery Server 时执行的提交验证器工作流会使用内容验证适配器。

内容验证 API 包括以下类：

- `ValidationAdapter` - 抽象类，您可以通过扩展该类创建您自己的内容验证适配器。请实现您自己的适配器以便根据需要验证或修改提交的内容。
- `ValidationContent` - 抽象类，您可以通过扩展该类创建您自己的 `ValidationContent` 对象。`ValidationContent` 对象包含用于标识已提交的内容的信息。

内容验证 API 的类位于 `com.sun.content.server.validation.adapter` 软件包中。有关这些类的其他信息，请参见位于 `$CDS_HOME/javadoc/validation/index.html` 的 Javadoc 工具的 HTML 输出。

5.1 一般处理流程

提交到 Content Delivery Server 的每项内容由 `$CDS_HOME/deployment/`*deployment-name*`/conf/SubmissionVerifierWorkflows.xml` 文件中定义的工作流处理。工作流由一系列步骤组成。在每个步骤中，该步骤指定的内容验证适配器将对内容二进制代码执行某种类型的处理。例如，一个步骤可能提供混淆处理，另一步骤可能为数字权限管理 (Digital Rights Management, DRM) 添加代码。有关创建内容验证工作流的信息，请参见 《Sun Java™ System Content Delivery Server 5.1 集成和配置指南》中的第 14.3 节 “创建工作流”。

对于工作流中的每个步骤， Content Delivery Server 调用为该步骤指定的内容验证适配器，并向其传递一个 `ValidationContent` 对象和一个 `Properties` 对象。`ValidationContent` 对象包含被提交内容的元数据和内容二进制代码。 `Properties` 对象包含工作流中为该步骤定义的参数。

对于工作流中的第一步，Content Delivery Server 创建初始 ValidationContent 对象。对于随后的每个步骤，前一步返回的 ValidationContent 对象将传递到下一步的内容验证适配器。

5.2 ValidationAdapter 类

ValidationAdapter 类处理元数据和内容二进制代码，并执行任何必要的转换。本节介绍需要实现的方法。

5.2.1 execute 方法

```
public abstract ValidationContent execute(ValidationContent content,
java.util.Properties properties) throws java.lang.Exception
```

执行工作流中的步骤时，Content Delivery Server 将调用 execute 方法。此方法的参数是上一个步骤中的 ValidationContent 对象以及在工作流步骤中指定的参数。您必须知道在工作流的上一个步骤中创建的 ValidationContent 对象的类型。例如，如果在工作流的第一步使用某个适配器，则该适配器必须准备从 Content Delivery Server 接收 InitialValidationContent 对象。

在实现此方法时，您必须处理传递的属性，并创建要返回的 ValidationContent 对象。根据需要处理元数据和内容二进制代码。例如，要对代码进行混淆处理，请调用所需的混淆器来转换内容二进制代码，并在 ValidationContent 对象中返回新的内容二进制代码。必须知道工作流下一步中预期的 ValidationContent 对象的类型并生成该对象类型。例如，如果工作流下一步预期接收定制的 ValidationObject，则此方法必须生成该定制的 ValidationObject。

如果您的适配器需要的信息可能更改，或者您不希望进行硬编码，则请为这些值创建属性文件。然后适配器可以通过作为参数传递的 Properties 对象访问文件中的信息。例如，如果您的适配器对代码进行混淆处理，则您可能需要一个属性来标识运行适配器的系统上混淆器的位置。创建的属性文件必须置于 $CDS_HOME/deployment/*deployment-name*/conf 目录中。还必须在 $CDS_HOME/deployment/*deployment-name*/conf/SubmissionVerifierAdapters.xml 文件中设置适配器的属性文件名，如第 5-3 页上的第 5.4 节 “使用内容验证 API”中所述。

5.2.2 returns 方法

```
public static java.lang.Class returns(java.lang.Class inputType)
throws java.lang.Exception
```

Content Delivery Server 调用 `returns` 方法以验证适配器能否处理要传递的 `ValidationContent` 对象类型。此方法返回的对象类型必须与 `execute` 方法返回的类型相同。例如，如果 `execute` 方法返回定制的 `ValidationContent` 对象，则 `returns` 方法必须返回相同类型的定制 `ValidationContent` 对象。

有关此方法的样例实现，请参见第 5-4 页上的第 5.5 节 “样例内容验证适配器”。

5.3 ValidationContent 类

`ValidationContent` 类是抽象类，可以扩展该类以创建定制的验证适配器。该类包含内容的元数据和二进制代码部分。如果工作流的某个后续步骤中的适配器需要额外的内容信息，请将该信息添加到从此类扩展而来的类中。

5.4 使用内容验证 API

`cdsapi.jar` 文件中提供了内容验证 API 的类。在编译适配器时，此文件必须位于类路径中。为方便起见，`$CDS_HOME/dist/cds/staging/jar` 目录中提供了所有 Content Delivery Server JAR 文件的副本。在编译所创建的适配器时，请在类路径中使用此缓冲区。

使适配器可用于 Content Delivery Server 取决于所使用的应用服务器，以及是否已部署了该服务器。要使适配器可用，请执行以下操作：

1. **为适配器创建 JAR 文件。**
2. **对于所有应用服务器，请将该 JAR 文件放在 `$CDS_HOME/dist/cds/lib/external` 目录中。**

   现在，该适配器将包含在所有以后的部署中。
3. **如果现有部署需要使用该适配器，请将该 JAR 文件放在每个部署的 `$CDS_HOME/deployment/`*deployment-name*`/lib/external` 目录中。**

   如果使用的是 WebLogic Server，则会为您处理类路径。

   如果使用的是 Sun Java System Application Server，则会为每个部署更新类路径：

   a. **在编辑 `$CDS_HOME/deployment/`*deployment-name*`/sun/domains/cdsdomain/config/domain.xml` 文件之前，先对其进行备份，以便从编辑期间可能引入的任何错误中恢复。**

   b. **编辑 `domain.xml` 并修改 `java-config` 元素，将 JAR 文件的绝对路径添加到 `classpath-suffix` 属性中。**

c. **保存所做的更改。**

4. **编辑** `$CDS_HOME/deployment/`*deployment-name*`/conf` **目录中的** `SubmissionVerifierAdapters.xml` **文件：**

   a. **为创建的适配器添加一个语句。**

   例如，如果创建名为 `MyValidationAdapter` 的适配器，且该适配器需要名为 `validation.properties` 的属性文件，请将以下语句添加到文件中：

```
<adapter id="MyValidationAdapter" name="sample.package.MyValidationAdapter"
propertyfile="validation.properties"/>
```

   b. **保存所做的更改。**

5. **编辑** `$CDS_HOME/deployment/`*deployment-name*`/conf` **目录中的** `SubmissionVerifierWorkflows.xml` **文件：**

   a. **在相应工作流中添加一个步骤，以执行所创建的适配器。**

   关于步骤元素的 `adapter` 属性值，请指定为适配器元素 （在步骤 a 中添加到 `SubmissionVerifierAdapters.xml` 文件的元素）的 `id` 属性提供的值。有关创建工作流的信息，请参见《Sun Java™ System Content Delivery Server 5.1 集成和配置指南》中的第 14.3 节 “创建工作流”。

   b. **保存所做的更改。**

6. **重新启动任何现有部署以识别新的 JAR 文件。**

5.5 样例内容验证适配器

以下代码示例是如何扩展 `ValidationAdapter` 类以实现您自己的验证适配器的样例。

编码样例 5-1 样例 `ValidationAdapter` 实现

```
import com.sun.content.server.validation.adapter.*;
import java.io.FileOutputStream;
import java.util.Properties;

public class ExportToFileValidationAdapter
extends ValidationAdapter
{
  public ValidationContent execute(
    ValidationContent content, Properties properties)
  throws Exception
  {
    // Export if the filename is specified
```

编码样例 5-1 样例 ValidationAdapter 实现 （续）

```
    String outFilename =
      properties.getProperty("ExportToFile.FileName");
    if (outFilename != null)
    {
      FileOutputStream fileOutStream = null;
      try
      {
        // get the first byte[] in the map
        // ignore the rest for this sample
        byte[] bytes = (byte[])
          content.getMimeBytesMap().values().iterator().next();

        // Write the byte[] to the output file.
        fileOutStream = new FileOutputStream(outFilename);
        fileOutStream.write(bytes);
        fileOutStream.flush();
        fileOutStream.close();
        fileOutStream = null;
      }
      finally
      {
        if (fileOutStream != null)
        {
          fileOutStream.flush();
          fileOutStream.close();
        }
      }
    }

    content.setStatus(ValidationContent.VALID);

    return content;
  }

  public static Class returns(Class inputType) throws Exception
  {
    if (!ValidationContent.class.isAssignableFrom(inputType))
      throw new Exception("Wrong input type to adapter.");
    return ValidationContent.class;
  }
}
```

第 6 章

用户配置 API

本章介绍了 Sun Java System Content Delivery Server 用户管理 API。可以使用此 API 来创建订户适配器，以便添加、删除、检索、更新、启用和禁用用户。此 API 还提供了一些方法，以便从 Content Delivery Server 数据库中检索与设备有关的信息。

用户配置 API 包含以下类和接口：

- `UserManager` - 抽象类，您可以扩展该类来创建自己的订户适配器。该类用于控制个人用户帐户的创建和状态。
- `User` - 接口，您可以实现该接口来管理每个用户的特定属性。
- `UserDeviceManager` - 可以实现该接口，以提供订户所使用的设备的相关设备信息和安全信息。可以使用此接口来防止订户只付费一次而将内容下载到几个设备上的情况。
- `UserDeviceModel` - 此接口提供 Content Delivery Server 数据库中的设备相关信息。

有关类和方法的其他信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 中的用户配置 API 的 Javadoc 工具的 HTML 输出。

6.1 UserManager 类

`UserManager` 类用于定义创建、删除或访问用户信息的方法。可以根据需要扩展此类并实现所有抽象方法，以便与订户数据库进行交互。要获取 Content Delivery Server 已知的设备的相关信息，可以使用 `getDeviceModel` 和 `getDeviceModelList` 方法。

有关未介绍的方法的信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 中的用户配置 API 的 Javadoc 工具的 HTML 输出。

6.1.1 doFormatMobileId 方法

```
protected abstract String doFormatMobileId(String mobileId);
```

doFormatMobileId 方法用于设置移动 ID 的格式，以符合订户数据库的要求。例如，如果数据库无法处理移动 ID 中的连字符，并且订户在 Content Delivery Server 中注册时输入的移动 ID 包含连字符，则可以实现该方法以删除这些特殊字符。

6.1.2 doFormatLoginId 方法

```
protected abstract String doFormatLoginId(String loginId);
```

doFormatLoginID 方法用于设置登录 ID 的格式，以符合订户数据库的要求。例如，如果数据库要求所有登录 ID 使用小写字母，并且订户在 Content Delivery Server 中注册时输入的 ID 包含大写字母，则可以实现该方法以将 ID 转换成小写字母。

6.1.3 doGetUserDeviceModel 方法

```
protected long doGetUserDeviceModel(User user, UserDeviceModel
model);
```

doGetUserDeviceModel 方法用于标识与订户关联的设备。当订户进行登录时，或者订户选择向未在 Content Delivery Server 中注册的用户赠送礼品或与其共享内容时，Content Delivery Server 将会调用该方法。

您必须返回以下值之一：

- 当前与订户关联的设备的手机 ID

  如果具有足够的信息来标识订户的设备，则会返回手机 ID。此 ID 是在系统中添加设备时 Content Delivery Server 指定的 ID。如果知道设备的用户代理模式，则可以调用 UserManager.getDeviceModel 方法并从返回的对象中获取手机 ID。要获取 Content Delivery Server 已知的设备列表，请调用 UserManager.getDeviceModelList 方法。可以按名称、型号或其他属性查找该列表，以确定所需的设备并获取手机 ID。如果返回的设备 ID 与传递的设备型号不同，则订户的用户配置会随新设备而更新。

- DEVICE_MODEL_IGNORE

  如果无法确定设备，或者要使用现有的设备 （如果有），则会返回该值。

- DEVICE_MODEL_UNKNOWN

  如果无法确定设备，或者要删除设备与订户之间的关联，则会返回该值。

6.2 User 接口

User 接口用于定义获取、设置或删除用户属性的通用方法。实现此接口和所有方法。有关其他信息，请参见 $CDS_HOME/javadoc/cdsapi/index.html 中的用户配置 API 的 Javadoc 工具的 HTML 输出。

6.3 UserDeviceManager 接口

UserDeviceManager 接口定义了一种方法，以根据设备连接时使用的 IP 地址访问设备的唯一 ID，例如，电子序列号 (Electronic Serial Number, ESN)。实现此接口及其方法。有关其他信息，请参见 $CDS_HOME/javadoc/cdsapi/index.html 中的用户配置 API 的 Javadoc 工具的 HTML 输出。

6.4 UserDeviceModel 接口

UserDeviceModel 接口定义了一些方法，以访问 Content Delivery Server 已知的设备的相关信息。您可能调用的方法会返回这种类型的对象。有关其他信息，请参见 $CDS_HOME/javadoc/cdsapi/index.html 中的用户配置 API 的 Javadoc 工具的 HTML 输出。

6.5 使用用户配置 API

cdsapi.jar 中提供了用户配置 API 的类。在编译适配器时，此文件必须位于类路径中。为方便起见， $CDS_HOME/dist/cds/staging/jar 目录中提供了所有 Content Delivery Server JAR 文件的副本。在编译所创建的适配器时，请在类路径中使用此缓冲区。

使适配器可用于 Content Delivery Server 取决于所使用的应用服务器，以及是否已部署了该服务器。要使适配器可用，请执行以下操作：

1. **为适配器创建 JAR 文件。**
2. **对于所有应用服务器，请将该 JAR 文件放在 $CDS_HOME/dist/cds/lib/external 目录中。**

   现在，该适配器将包含在所有以后的部署中。

3. **如果现有部署需要使用该适配器，请将该 JAR 文件放在每个部署的 `$CDS_HOME/deployment/`*deployment-name*`/lib/external` 目录中。**

   如果使用的是 WebLogic Server，则会为您处理类路径。

   如果使用的是 Sun Java System Application Server，则会为每个部署更新类路径：

   a. **在编辑 `$CDS_HOME/deployment/`*deployment-name*`/sun/domains/cdsdomain/config/domain.xml` 文件之前，先对其进行备份，以便从可能在编辑期间引入的任何错误中恢复。**

   b. **编辑 `domain.xml` 并修改 `java-config` 元素，将 JAR 文件的绝对路径添加到 `classpath-suffix` 属性中。**

   c. **保存所做的更改。**

4. **编辑 `$CDS_HOME/deployment/`*deployment-name*`/conf` 目录中的 `security.config` 文件：**

   a. **将 `module.security.subscriber.usermanager` 属性设置为 `User` 接口实现的类名，例如：**

   module.security.subscriber.usermanager=
   com.sun.content.server.vending.security.user.SubscriberImpl

   属性名称和值必须在同一行中。此样例语句占两行是为了适应页面大小。

   b. **保存所做的更改。**

5. **重新启动任何现有部署以识别新的 JAR 文件。**

6.6 用户管理器 API 的样例实现

本节中的代码样例提供了一个使用用户配置 API 创建的订户适配器示例。在 Content Delivery Server 中创建订户帐户时，该实现将在本地文件系统中对用户信息进行编索并存储这些信息。

6.6.1 SampleUserImpl.java

以下代码是 `User` 接口的样例实现。此接口包括 Content Delivery Server 用于用户配置的字段。如果特定实现具有其他字段，请添加所需的方法以获取和设置这些值。

编码样例 6-1　　样例 User 实现

```
package com.wireless.adapter;

import com.sun.content.server.service.security.User;
import java.util.Date;
import java.util.Hashtable;
import java.io.Serializable;

public class SampleUserImpl implements User, Serializable {

    long uid = -1L;

    private String loginId;
    private String firstName;
    private String lastName;
    private String email;
    private String middleName;
    private String gender;
    private String street1;
    private String street2;
    private String postalCode;
    private String city;
    private String state;
    private String country;
    private String phone;
    private String salutation;
    private boolean enabled;
    String password;
    private String uniqueDeviceId;
    private String mobileId;
    private boolean prepay;

    public SampleUserImpl() {
    }

    public String getLoginId() { return loginId; }
    public String getFirstName() { return firstName; }
    public String getLastName() { return lastName; }
    public String getEmail() { return email; }
    public void setLoginId(String loginId) { this.loginId = loginId; }
    public void setFirstName(String firstName) { this.firstName = firstName; }
    public void setLastName(String lastName) { this.lastName = lastName; }
    public void setEmail(String email) { this.email = email; }
    public void setMiddleName(String middleName) {
        this.middleName = middleName;
    }
```

编码样例 6-1 样例 User 实现

```
    public String getMiddleName() { return middleName; }
    public void setGender(String gender) { this.gender = gender; }
    public String getGender() { return gender; }
    public String getStreet1() { return street1; }
    public void setStreet1(String street1) { this.street1 = street1; }
    public String getStreet2() { return street2; }
    public void setStreet2(String street2) { this.street2 = street2; }
    public String getPostalCode() { return postalCode; }
    public void setPostalCode(String postalCode) {
        this.postalCode = postalCode;
    }
    public String getCity() { return city; }
    public void setCity(String city) { this.city = city; }
    public String getState() { return state; }
    public void setState(String state) { this.state = state; }
    public String getCountry() { return country; }
    public void setCountry(String country) { this.country = country; }
    public String getPhone() { return phone; }
    public void setPhone(String phone) { this.phone = phone; }
    public String getSalutation() { return salutation; }
    public void setSalutation(String salutation) {
        this.salutation = salutation;
    }
    public boolean isEnabled() { return enabled; }
    public void setIsEnabled(boolean enabled) { this.enabled = enabled; }
    public String getPassword() {
        // we will not disclose password !
        return null;
    }
    public void setPassword(String password) { this.password = password; }
    public String getUniqueDeviceId() { return uniqueDeviceId; }
    public void setUniqueDeviceId(String uniqueDeviceId) {
        this.uniqueDeviceId = uniqueDeviceId;
    }
    public String getMobileId() { return mobileId; }
    public void setMobileId(String mobileId) { this.mobileId = mobileId; }
    public boolean isPrepay() { return prepay; }
    public void setIsPrepay(boolean prepay) { this.prepay = prepay; }

    // deprecated methods
    public Date getCreateDate() { return null; }
    public void setCreateDate(Date date) {}
    public Date getLastLogin() { return null; }
    public void setLastLogin(Date date) {}
    public Object getAttribute(String name) { return null; }
    public Object getAttribute(String name, Object def) { return null; }
    public Hashtable getAttributes() { return null; }
```

编码样例 6-1 样例 User 实现

```
    public void setHasLoggedIn(boolean b) {}
    public boolean hasLoggedIn() { return false; }
    public void setAttribute(String name, Object val) {}
    public void setAttributes(Hashtable h) {}
    public boolean isConfirmed() { return false; }
    public void updateLastLogin() {}
    public Date getActivateDate() { return null; }
    public void setActivateDate(Date date) {}
    public Date getDeActivateDate() { return null; }
    public void setDeActivateDate(Date date) {}

}
```

6.6.2 SampleUserManagerImpl.java

以下代码是 UserManager 类的样例扩展。

编码样例 6-2 样例 UserManager 实现

```
package com.wireless.adapter;

import com.sun.content.server.service.security.*;
import com.sun.content.server.foundation.log.LogCategory;
import com.sun.content.server.service.security.util.*;
import java.io.*;
import java.util.*;
import java.util.concurrent.atomic.*;

/**
 * A sample user manager for Content Delivery Server.
 *
 * <p>
 * The sample user manager indexes and stores user information on a
 * local file system. The implementation is rudimentary, and doesn't support
 * concurrent file modification of the database, so only one server should
 * be sharing the user data.
 *
 * <p>
 * The user manager is provided strictly for educational and experimental
 * purposes, it has not passed any testing.
 */
public class SampleUserManagerImpl extends UserManager{

    // you can change these constants
    private final static String FS_ROOT = "/var/tmp/cds_user_db";
    private final static String FS_INDEX_LOGIN = "login.idx";
```

编码样例 6-2 样例 UserManager 实现

```
    private final static String FS_INDEX_UNIQUE = "unique.idx";
    private final static String FS_INDEX_MOBILE = "mobile.idx";
    private final static String FS_INDEX = "master.idx";
    private final static String DS_KEY = "____sum___timestamp";
    private final static long LDS_KEY = -1L;
    private final static String EXT = ".act";

    private static Map<String,Long> indexLogin;
    private static Map<String,Long> indexUnique;
    private static Map<String,Long> indexMobile;
    private static Map<Long,String[]> index;
    private static Object idxLock = new Object();

    private static AtomicLong seq = new AtomicLong();

    private static boolean dirty;
    private static Object dirtyLock = new Object();

    private static LogCategory log = LogCategory.getLog();

    static {
        init();
    }

    private static void init() {

        File fs = new File(FS_ROOT);

        if (!fs.exists()) {
            if (!fs.mkdirs()) {
                throw new RuntimeException("Can not create directories"+
                        " for user database " + fs.getAbsolutePath());
            }
        }

        File f = new File(fs, FS_INDEX_LOGIN);
        indexLogin = readMap(f);
        f = new File(fs, FS_INDEX_UNIQUE);
        indexUnique = readMap(f);
        f = new File(fs, FS_INDEX_MOBILE);
        indexMobile = readMap(f);
        f = new File(fs, FS_INDEX);
        index = readMap(f);

        if (index == null) {
            rebuildMasterIndex();
        }
```

编码样例 6-2 样例 UserManager 实现

```
        long masterStamp = Long.parseLong(index.get(LDS_KEY)[0]);
        if (indexLogin != null && indexLogin.get(DS_KEY) < masterStamp) {
            indexLogin = null;
        }
        if (indexUnique != null && indexUnique.get(DS_KEY) < masterStamp) {
            indexUnique = null;
        }
        if (indexMobile != null && indexMobile.get(DS_KEY) < masterStamp) {
            indexMobile = null;
        }

        boolean needIL = false;
        boolean needIU = false;
        boolean needIM = false;

        if (indexLogin == null) {
            log.info("SUM: will rebuild login index");
            needIL = true;
            indexLogin = new HashMap<String,Long>();
        }
        if (indexUnique == null) {
            log.info("SUM: will rebuild unique index");
            needIU = true;
            indexUnique = new HashMap<String,Long>();
        }
        if (indexMobile == null) {
            log.info("SUM: will rebuild mobile index");
            needIM = true;
            indexMobile = new HashMap<String,Long>();
        }

        long maxVal = 0L;

        for (Map.Entry<Long,String[]> entry : index.entrySet()) {

            Long key = entry.getKey();

            if (key.equals(LDS_KEY)) { continue; }

            String [] data = entry.getValue();

            if (needIL) {
                indexLogin.put(data[1], key);
            }
            if (needIU) {
                indexUnique.put(data[2], key);
```

编码样例 6-2 样例 UserManager 实现

```
            }
            if (needIM) {
                indexMobile.put(data[3], key);
            }

            if (maxVal < key) { maxVal = key; }
        }

        seq.set(maxVal);

        log.info("SUM: initialized, index is "+ index.size()+" entries");

        dirty = true;

        Runnable r = new Updater();
        (new Thread(r)).start();

    }

    private static void rebuildMasterIndex() {

        try {
            log.info("SUM: rebuilding master index");
            File fs = new File(FS_ROOT);

            index = new HashMap<Long,String[]>();

            File [] lst = fs.listFiles(new FilenameFilter() {
                public boolean accept(File dir, String name) {
                    return name.endsWith(EXT);
                } });

            for (File file : lst) {
                SampleUserImpl user = readUser(file, false);
                if (user == null) { continue; }
                if (seq.get() <= user.uid) { seq.set(user.uid + 1L); }
                index.put(user.uid, new String[]{
                        file.getName(),
                        user.getLoginId(),
                        user.getUniqueDeviceId(),
                        user.getMobileId()});
            }

            index.put(LDS_KEY,
                    new String[]{String.valueOf(System.currentTimeMillis())});
        } catch (Exception e) {
            e.printStackTrace();
```

编码样例 6-2　　样例 UserManager 实现

```
            throw new RuntimeException(e);
        }
    }

    private static SampleUserImpl readUser(File f,
            boolean reThrow) throws Exception {

        try {
            FileInputStream fis = new FileInputStream(f);
            ObjectInputStream ois = new ObjectInputStream(fis);
            SampleUserImpl user = (SampleUserImpl)ois.readObject();
            ois.close();
            fis.close();
            return user;
        } catch (Exception e) {
            if (reThrow) { throw e; }
            log.warn("error reading file "+f.getAbsolutePath(), e);
        }
        return null;

    }

    private static Map readMap(File f) {
        try {

            FileInputStream fis = new FileInputStream(f);
            ObjectInputStream ois = new ObjectInputStream(fis);
            Map map = (Map)ois.readObject();
            ois.close();
            fis.close();
            return map;

        } catch (Exception e) {
            log.warn("error reading file "+f.getAbsolutePath(), e);
        }
        return null;
    }

    public SampleUserManagerImpl() {
    }

    public boolean doAddUser(User _user) throws UserProfileResourceException {
        SampleUserImpl user = (SampleUserImpl)_user;

        synchronized (user) {

            if (user.uid != -1L) {
```

编码样例 6-2 样例 UserManager 实现

```
                throw new UserProfileResourceException("the user is already"+
                        " added");
            }

            user.uid = seq.incrementAndGet();

            doUpdateUser(_user);
        }

        return true;
    }

    protected User doGetUserInstance() {
        return new SampleUserImpl();
    }

    protected User doGetUserByUniqueDeviceId(String id)
            throws UserProfileResourceException {
        return getUserFromIndex(id, indexUnique);
    }

    protected User doGetUserByMobileId(String id)
            throws UserProfileResourceException {
        return getUserFromIndex(id, indexMobile);
    }

    protected User doGetUserByLoginId(String id)
            throws UserProfileResourceException {
        return getUserFromIndex(id, indexLogin);
    }

    protected boolean doAuthenticatePassword(User _user, String pwd) {
        return pwd.equals(((SampleUserImpl)_user).password);
    }

    protected void doEnableUser(User _user)
            throws UserProfileResourceException {

        SampleUserImpl user = (SampleUserImpl)_user;
        if (user.isEnabled()) { return; }
        user.setIsEnabled(true);
        doUpdateUser(_user);
    }

    protected void doDisableUser(User _user)
            throws UserProfileResourceException {
```

编码样例 6-2 样例 UserManager 实现

```
        SampleUserImpl user = (SampleUserImpl)_user;
        if (!user.isEnabled()) { return; }
        user.setIsEnabled(false);
        doUpdateUser(_user);
    }

    protected boolean doRemoveUser(User _user)
            throws UserProfileResourceException {

        synchronized (idxLock) {
            SampleUserImpl user = (SampleUserImpl)_user;
            String [] data = index.get(user.uid);
            if (data == null) { return false; }
            index.remove(user.uid);
            indexLogin.remove(data[1]);
            indexUnique.remove(data[2]);
            indexMobile.remove(data[3]);
            File f = new File(FS_ROOT, data[0]);
            f.delete();
            user.uid = -1L;
        }
        synchronized (dirtyLock) {
            dirty = true;
        }

        return true;
    }

    protected boolean doUpdateUser(User _user)
            throws UserProfileResourceException {

        SampleUserImpl user = (SampleUserImpl)_user;

        if (user.uid == -1L) { return false; }

        try {

            File fs = new File(FS_ROOT);

            String [] data = index.get(user.uid);
            if (data == null) {
                File tc = File.createTempFile("sum-", EXT, fs);
                data = new String[]{
                    tc.getName(), user.getLoginId(),
                    user.getUniqueDeviceId(), user.getMobileId()
                };
            }
```

编码样例 6-2 样例 UserManager 实现

```
            // dump out user data

            File f = new File(fs, data[0]);
            FileOutputStream fos = new FileOutputStream(f);
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(user);
            oos.close();
            fos.close();

            synchronized (idxLock) {
                index.put(user.uid, data);
                indexLogin.remove(data[1]);
                indexLogin.put(user.getLoginId(), user.uid);
                indexUnique.remove(data[2]);
                indexUnique.put(user.getUniqueDeviceId(), user.uid);
                indexMobile.remove(data[3]);
                indexMobile.put(user.getMobileId(), user.uid);
            }

            synchronized (dirtyLock) {
                dirty = true;
            }


        } catch (Exception e) {
            e.printStackTrace();
            throw new UserProfileResourceException(e);
        }

        return true;

    }

    private User getUserFromIndex(String id,
            Map<String,Long> index) throws UserProfileResourceException {

        try {
            Long pk = index.get(id);
            if (pk == null) { return null; }
            return getUserByPK(pk);
        } catch (Exception e) {
            e.printStackTrace();
            throw new UserProfileResourceException(e);
        }
    }
```

编码样例 6-2　　样例 UserManager 实现

```
    private User getUserByPK(Long pk) throws Exception {
        String [] data = index.get(pk);
        if (data == null) { return null; }
        File f = new File(FS_ROOT, data[0]);
        return readUser(f, true);
    }

    protected String doFormatMobileId(String id) {
        return id.trim();
    }

    protected String doFormatLoginId(String id) {
        return id.trim();
    }

    static class Updater implements Runnable {

        public void run() {

            Thread.currentThread().setName("SUM-Updater");

            log.info("SUM: updater thread initialized");

            while (true) {

                try {
                    Thread.sleep(2000);
                } catch (Exception e) {}

                synchronized (dirtyLock) {
                    if (!dirty) { continue; }
                    dirty = false;
                }
                // the maps are dirty, well...

                try {

                    long stamp = System.currentTimeMillis();

                    Map<Long,String[]> _index = new HashMap<Long,String[]>();
                    Map<String,Long> _login = new HashMap<String,Long>();
                    Map<String,Long> _unique = new HashMap<String,Long>();
                    Map<String,Long> _mobile = new HashMap<String,Long>();

                    synchronized (idxLock) {
                       index.put(LDS_KEY, new String[]{String.valueOf(stamp)});
                        indexLogin.put(DS_KEY, stamp);
```

编码样例 6-2 样例 UserManager 实现

```
                        indexUnique.put(DS_KEY, stamp);
                        indexMobile.put(DS_KEY, stamp);

                        _index.putAll(index);
                        _login.putAll(indexLogin);
                        _unique.putAll(indexUnique);
                        _mobile.putAll(indexMobile);
                    }

                    File f = new File(FS_ROOT, FS_INDEX);
                    FileOutputStream fos = new FileOutputStream(f);
                    ObjectOutputStream oos = new ObjectOutputStream(fos);
                    oos.writeObject(_index);
                    oos.close();
                    fos.close();

                    f = new File(FS_ROOT, FS_INDEX_LOGIN);
                    fos = new FileOutputStream(f);
                    oos = new ObjectOutputStream(fos);
                    oos.writeObject(_login);
                    oos.close();
                    fos.close();

                    f = new File(FS_ROOT, FS_INDEX_UNIQUE);
                    fos = new FileOutputStream(f);
                    oos = new ObjectOutputStream(fos);
                    oos.writeObject(_unique);
                    oos.close();
                    fos.close();

                    f = new File(FS_ROOT, FS_INDEX_MOBILE);
                    fos = new FileOutputStream(f);
                    oos = new ObjectOutputStream(fos);
                    oos.writeObject(_mobile);
                    oos.close();
                    fos.close();

                    log.debug("SUM: indices synced to fs");

                } catch (Exception e) {
                    e.printStackTrace();
                }

            }

        }
```

编码样例 6-2 样例 UserManager 实现

```
    }

}
```

6.6.3 使用该样例适配器

如果要试验该代码，您可以在测试部署中运行该样例适配器。第一次部署该适配器时，Vending Manager 中不能有任何订户帐户。

要使用该样例适配器，请执行以下操作：

1. **如果 Vending Manager 正在运行，请将其停止。**

   必须在部署了 Vending Manager 的服务器上运行该订户适配器。

2. **在目录中创建** `SampleUserImpl` **和** `SampleUserManagerImpl` **类，例如** `/tmp/sum`**。**

3. **在包含刚创建的类文件的目录中，按顺序执行以下命令。**

   以单行代码的形式输入每个命令。

   ```
   $ mkdir bin
   ```

   ```
   $ javac -d bin -classpath
   $CDS_HOME/dist/cds/lib/cdslib/cdsapi.jar:$CDS_HOME/dist/cds/lib/c
   dslib/foundation.jar SampleUserImpl.java
   SampleUserManagerImpl.java && ( cd bin && jar cf ../sum.jar .)
   ```

   ```
   $ cp sum.jar $CDS_HOME/deployment/deployment-name/lib/external/
   ```

4. **编辑** `$CDS_HOME/deployment/`*deployment-name*`/conf/security.config` **文件。**

   将 `module.security.subscriber.usermanager` 属性设置为 `com.wireless.adapter.SampleUserManagerImpl`。

5. **启动 Vending Manager。**

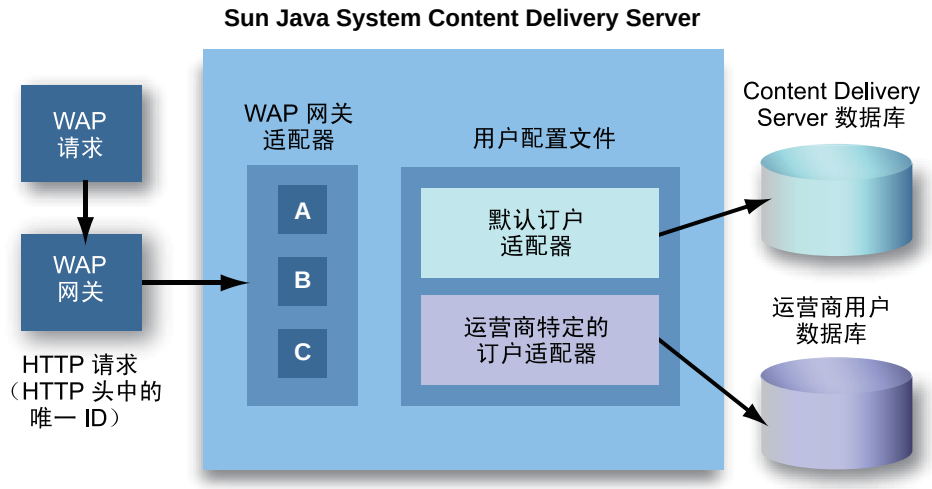
第 7 章

WAP 网关 API

本章介绍了 Sun Java System Content Delivery Server WAP 网关 API。此 API 从 HTTP 头中检索 MSISDN、设备配置文件以及其他属性。基于用户名和密码的身份验证在 WAP 网关集成中不是必需的。

图 7-1 是与 API 和访问点进行交互的通用系统组件的简单表示。其中还包括一些未与 API 进行交互、却是全面理解体系结构所需的组件。

图 7-1 WAP 网关适配器体系结构



运营商特定的订户适配器是用户配置 API 的实现

设备通过 WAP 网关将 WAP 请求发送到 Content Delivery Server。 Content Delivery Server 中的 WAP 网关适配器从 WAP 网关解析 HTTP 头并获得 MSISDN。订户适配器使用 MSISDN 或唯一 ID 来访问用户配置。

有关所有类和方法的信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 中的 WAP 网关 API 的 Javadoc 工具的 HTML 输出。

7.1 WAPGatewayAdapter 类

公用的抽象 `WAPGatewayAdapter` 类定义从 HTTP 头获得 MSISDN 以及唯一设备 ID 的方法。它还用于定义检查是否支持或可以实现此方法的方法。

可以扩展此类并实现所有抽象方法。有关其他信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 中的 WAP 网关 API 的 Javadoc 工具的 HTML 输出。

7.2 使用 WAP 网关 API

Content Delivery Server 为以下 WAP 网关提供了 API 实现。

- Nokia Activ Server 2.0.1
- Nokia Artus WAP 网关
- Openwave WAP 网关

必须配置 WAP 网关，以将 MSISDN 或唯一设备 ID 转发到 Content Delivery Server。`cdsapi.jar` 中提供了 `WAPGatewayAdapter` 类。为方便起见，`$CDS_HOME/dist/cds/staging/jar` 目录中提供了所有 Content Delivery Server JAR 文件的副本。

要为任何其他 WAP 网关注册一个新类以扩展 `WAPGatewayAdapter` 类，请将类文件名添加到 `$CDS_HOME/deployment/`*`deployment-name`*`/conf` 目录中的 `wapgateway.config` 文件中。下面的示例语句说明了如何在 Content Delivery Server 中注册 Nokia Activ Server 2.0.1 适配器。

```
module.gateway.id=
com.sun.content.server.common.gateway.nokia.NokiaActivServerWAPGateway
```

使适配器可用于 Content Delivery Server 取决于所使用的应用服务器，以及是否已部署了该服务器。要使适配器可用，请执行以下操作：

1. **为适配器创建 JAR 文件。**
2. **对于所有应用服务器，请将该 JAR 文件放在 `$CDS_HOME/dist/cds/lib/external` 目录中。**

   现在，该适配器将包含在所有以后的部署中。

3. **如果现有部署需要使用该适配器，请将该 JAR 文件放在每个部署的** `$CDS_HOME/deployment/`*deployment-name*`/lib/external` **目录中。**

   如果使用的是 WebLogic Server，则会为您处理类路径。

   如果使用的是 Sun Java System Application Server，则会为每个部署更新类路径：

   a. **在编辑** `$CDS_HOME/deployment/`*deployment-name*`/sun/domains/cdsdomain/config/domain.xml` **文件之前，先对其进行备份，以便从编辑期间可能引入的任何错误中恢复。**

   b. **编辑** `domain.xml` **并修改** `java-config` **元素，将 JAR 文件的绝对路径添加到** `classpath-suffix` **属性中。**

   c. **保存所做的更改。**

4. **重新启动任何现有部署以识别新的 JAR 文件。**

7.3 样例 WAP 网关适配器

以下代码示例显示了用于扩展 `WAPGatewayAdapter` 类的 `SampleWAPGateway` 的适配器的伪代码。

编码样例 7-1 使用 `WAPGatewayAdapter` 类的示例

```
package com.sun.content.server.service.gateway.sample;

import com.sun.content.server.service.gateway.WAPGatewayAdapter;
import com.sun.content.server.service.gateway.WAPGatewayException;
import javax.servlet.http.HttpServletRequest;

public class SampleWAPGateway extends WAPGatewayAdapter
{
  /* Method to check if the passed method is implemented in this
   * class or not. */
  public boolean doHandle(String method) throws WAPGatewayException
  {
    if (method.equals("getMSISDN"))
      return true;
    return false;
  }

  /* Gets the MSISDN from the header and returns as a string. */
  public String getMSISDN(HttpServletRequest request)
  {
    return request.getHeader("<key to retrieve>");
```

编码样例 7-1 使用 WAPGatewayAdapter 类的示例 （续）

```
  }

  /* This method is not implemented. */
  public String getUniqueId(HttpServletRequest req)
throws WAPGatewayException
  {
    throw new WAPGatewayException("This method is not implemented");
  }
}
```

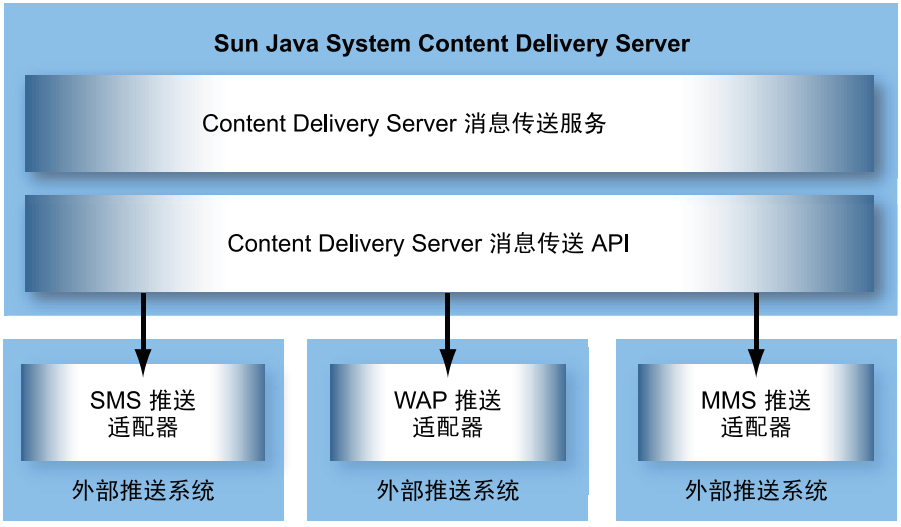
第 8 章

消息传送 API

本章介绍 Sun Java System Content Delivery Server 消息传送 API。消息传送 API 为运营商或应用程序供应商提供了一种机制，通过创建推送适配器将自身的 WAP、SMS 和 MMS 推送实现与 Content Delivery Server 进行集成。消息传送 API 用于创建推送发送器适配器和推送监听器适配器。

下图说明了消息传送 API 的高层体系结构以及使用此 API 的其他组件。

图 8-1 消息传送 API 的体系结构



外部推送系统可以与 Content Delivery Server 进行交互或者从中接收推送消息。

有关类和方法的信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 中的消息传送 API 的 Javadoc 工具的 HTML 输出。

8.1 PushMsgSender 接口

PushMsgSender 接口声明推送消息发送器实现所需的方法。此接口在 Content Delivery Server 发送推送消息时由 Content Delivery Server 消息传送服务调用。应实现此接口和所有方法。有关其他信息，请参见 $CDS_HOME/javadoc/cdsapi/index.html 中的消息传送 API 的 Javadoc 工具的 HTML 输出。

8.2 PushMsgListener 接口

PushMsgListener 接口声明推送消息监听器实现所需的方法。应实现此接口和所有方法。有关其他信息，请参见 $CDS_HOME/javadoc/cdsapi/index.html 中的消息传送 API 的 Javadoc 工具的 HTML 输出。

8.3 PushMessage 类

PushMessage 是由 Content Delivery Server 生成的所有不同类型的推送消息的基类。用于 MMS 消息的 MMSPushMessage 类将扩展 PushMessage 类。

在消息构建期间，所有 set 方法由 Content Delivery Server 使用。所有 get 方法可以由推送消息的发送器实现使用。

8.4 SMSMessage 类

SMSMessage 类已过时。以下方法现在是 PushMessage 类的一部分：

- getContentType
- getMIMEType
- setContentType
- setMIMEType

要获取并设置内容名称或下载 URL，请使用 PushMessage 类中的 getAttribute 和 setAttribute 方法，并且在 PushConstants 中定义相应的常量。有关详细信息，请参见 $CDS_HOME/javadoc/cdsapi/index.html 中这些类的 Javadoc 工具的输出。

8.5 WapPushMessage 类

WapPushMessage 类已过时。要获取并设置下载 URL，请使用 PushMessage 类中的 getAttribute 和 setAttribute 方法，并且在 PushConstants 中定义相应的常量。有关详细信息，请参见 $CDS_HOME/javadoc/cdsapi/index.html 中这些类的 Javadoc 工具的输出。

8.6 SMTPMessage 类

SMTPMessage 类已过时。要获取并设置主题或消息 “发件人” 地址，请使用 PushMessage 类中的 getAttribute 和 setAttribute 方法，并且在 PushConstants 中定义相应的常量。有关详细信息，请参见 $CDS_HOME/javadoc/cdsapi/index.html 中这些类的 Javadoc 工具的输出。

8.7 ContentSlide 类

ContentSlide 类用于存储来自 MMS 推送消息的二进制数据。该二进制数据可以具有 MIME 类型并具有与之关联的唯一 ID。在消息构建期间，所有 set 方法由服务器使用。所有 get 方法可以由推送消息的实现使用。有关详细信息，请参见 $CDS_HOME/javadoc/cdsapi/index.html 中这些类的 Javadoc 工具的输出。

8.8 MMSSlide 类

MMSSlide 类是 ContentSlide 对象的包装，用于构建 MMS 推送消息。在消息构建期间，所有 set 方法由 Content Delivery Server 使用。所有 get 方法可以由推送消息的实现使用。有关详细信息，请参见 $CDS_HOME/javadoc/cdsapi/index.html 中这些类的 Javadoc 工具的输出。

8.9 MMSPushMessage 类

MMSPushMessage 类扩展了 PushMessage 并表示了 MMS 推送消息。在消息构建期间，所有 set 方法由 Content Delivery Server 使用。所有 get 方法可以由推送消息的实现使用。此类包含 “发件人” 地址、“收件人” 地址、 MMSC 相关数据、用户代理以及任何同步多媒体集成语言 (Synchronized Multimedia Integration Language, SMIL) 数据 （如果可用）。同时还封装了 MMSSlide 对象。有关详细信息，请参见 $CDS_HOME/javadoc/cdsapi/index.html 中这些类的 Javadoc 工具的输出。

8.10 MMSSender 接口

MMSSender 接口声明用于发送 MMS 消息的方法。如果要支持 MMS 消息，则此接口必须在集成过程中由供应商特定的 MMSC 实现。有关详细信息，请参见 $CDS_HOME/javadoc/cdsapi/index.html 中这些类的 Javadoc 工具的输出。

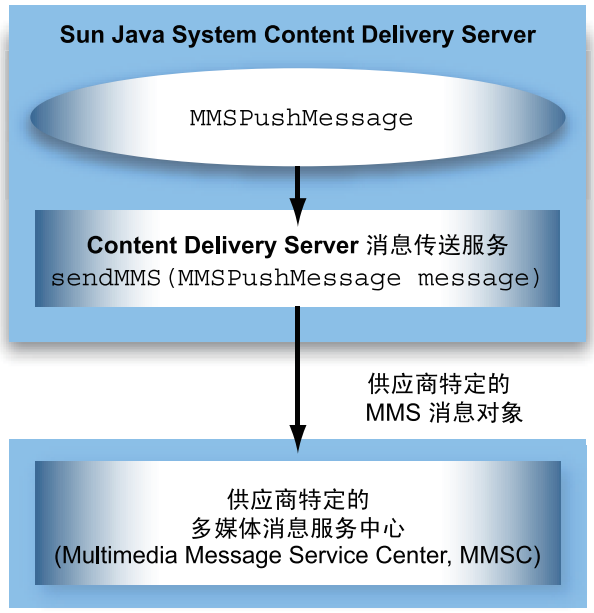
8.10.1 sendMMS

```
sendMMS(com.sun.content.server.server.messaging.message.MMSPushMessage message)
```

sendMMS 方法封装用于发送 MMS 消息的功能。此方法由 Content Delivery Server 消息传送服务在从 Content Delivery Server 接收 MMSPushMessage 时调用。

Content Delivery Server 将 MMSPushMessage 对象发送到 Content Delivery Server 消息传送服务。消息传送服务使用 MsgService.properties 文件中的 mms.senderclass 属性值标识提供 MMSSender.sendMMS 的供应商特定实现的类的全限定名称。 sendMMS 方法实现中的代码将 MSSPushMessage 对象转换为 MMS 消息对象的供应商特定版本并将其发送到供应商特定 MMSC 进行处理。下图显示了此过程。

图 8-2 发送 MMS 消息的进程流



要实现 `MMSSender.sendMMS` 方法的供应商特定版本，须包含以下项：

1. 连接到供应商特定的 MMSC。
2. 实现 `MMSSender.sendMMS` 方法以执行下列任务：
   - 将传递到 Content Delivery Server 消息传送服务的 `MMSPushMessage` 对象转换为供应商特定的 MMS 消息对象。
   - 将新的消息对象发送到供应商特定的 MMSC。
   - 在提交 MMS 消息时，通过从 MMSC 接收的数据构建 `PushResponse` 对象并返回该对象。

有关详细信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 中这些类的 Javadoc 工具的输出。

8.11 PushResponse 类

PushResponse 类是由外部推送服务生成的所有不同类型的推送响应的基类。所有 set 方法由推送消息的发送器实现使用。所有 get 方法可以由 Content Delivery Server 使用以便将推送消息记录到数据库中。有关详细信息，请参见 $CDS_HOME/javadoc/cdsapi/index.html 中这些类的 Javadoc 工具的输出。

8.12 PushConstants 类

PushConstants 类包含所有推送服务支持的常量。特定推送服务实现将这些常量与从 PushMessage 对象接收的值进行比较。有关所定义常量的信息，请参见 $CDS_HOME/javadoc/cdsapi/index.html 中的 Javadoc 工具的输出。

8.13 使用消息传送 API

推送发送器适配器必须实现 PushMsgSender 接口。部署期间，实现类需要在 Content Delivery Server 中注册。 XML 文件用于注册。此 XML 文件位于 $CDS_HOME/deployment/*deployment-name*/conf 目录中，且名为 pushsenderfactory.xml。

以下示例显示了此文件的结构。

编码样例 8-1 样例 pushsenderfactory.xml 文件

```
<pushmsgsenderset>
  <pushmsgsender0 class =
"com.sun.content.server.messagingservice.msgserver.push.TestSMSPushMsgSenderImpl"
    protocol="sms"/>
  <pushmsgsender1 class =
"com.sun.content.server.messagingservice.msgserver.push.WAPPushMsgSender"
    protocol="wap"/>
  <pushmsgsender2 class =
"com.sun.content.server.messagingservice.msgserver.push.SMTPPushMsgSender"
    protocol="smtp"/>
  <pushmsgsender3 class =
"com.sun.content.server.messagingservice.msgserver.push.MMSPushMsgSender"
  protocol="mms"/>
</pushmsgsenderset>
```

此文件中注册了四个适配器。请指定适配器支持的全限定类名和协议。例如，如果适配器用于 SMS 推送，则协议为 sms。确保在类路径中设置了适配器类和相关的类。

如果使用的是 MMS 推送发送器适配器的默认实现 MMSPushMsgSender，则必须实现 MMSSender 接口。请将 MsgServices.properties 文件中的 mms.senderclass 属性设置为类的全限定名称。该类位于 $CDS_HOME/deployment/*deployment-name*/conf 目录中。

推送监听器适配器实现了 PushMsgListener 接口。实现类需要在 Content Delivery Server 中注册。 XML 文件用于注册。此 XML 文件位于 $CDS_HOME/deployment/*deployment-name*/conf 目录中，且名为 pushlistenerfactory.xml。

以下示例显示了此文件的结构。

编码样例 8-2 样例 pushlistenerfactory.xml 文件

```
<pushmsglistenerset>
    <pushmsglistener0 class=
"com.sun.content.server.messagingservice.msgserver.protocol.cimd2.CIMD2PushMsgListener"
 protocol="sms"/>
</pushmsglistenerset>
```

第 9 章

确认服务 API

通过使用 Sun Java System Content Delivery Server 确认服务 API， Content Delivery Server 可以处理将内容下载到设备后发送的确认消息。只有在下列情况下才需要确认服务适配器：为 Content Delivery Server 定义了能够发送确认消息的设备，或者使用可发送确认消息的服务。

Content Delivery Server 支持以下类型的确认消息：

- 通过多媒体消息服务中心 (Multimedia Messaging Service Center, MMSC) 使用 MMS 发送的消息
- 使用用户数据报协议 (User Datagram Protocol, UDP) 直接发送到 Content Delivery Server 的消息

确认服务 API 包括以下类：

- `ConfirmServiceAdapter` - 抽象类，可以扩展该类以连接到传送确认消息的外部实体并监听消息。
- `ConfirmResponse` - 包含收到的确认信息的类。
- `ConfirmServiceException` - 确认服务 API 抛出的异常。

有关这些类的其他信息，请参见位于 `$CDS_HOME/javadoc/cdsapi/index.html` 的 Javadoc 工具的 HTML 输出。

9.1 一般处理流程

Content Delivery Server 可以将多媒体消息中的内容发送到支持 MMS 标准的设备。当设备收到 MMS 消息内容时，将通过 MMSC 返回确认消息。使用确认服务 API 编写的确认服务适配器用于设置 Content Delivery Server 与 MMSC 间的连接，以及处理来自 MMSC 的确认消息。

如果将系统设置为支持将 UDP 消息直接发送到 Content Delivery Server 的服务或设备，则确认服务适配器会将 Content Delivery Server 设置为监听并处理这些确认消息。

9.2 ConfirmServiceAdapter 类

`ConfirmServiceAdapter` 类建立连接以接收确认消息，并将收到的消息传送到 Content Delivery Server。扩展 `ConfirmServiceAdapter` 以创建系统的确认服务适配器。

`ConfirmServiceAdapter` 类位于 `com.sun.content.server.confirmservice` 软件包中。

9.2.1 connect 方法

```
public abstract boolean connect() throws ConnectionFailedException
```

可以使用该方法将 Content Delivery Server 连接到 MMSC，或者将 Content Delivery Server 设置为接收 UDP 消息。

9.2.2 init 方法

```
public void init(Properties properties) throws
ConfirmServiceException
```

如果需要，可以使用此方法初始化该类的实现。该方法的实现是可选的。

9.2.3 listen 方法

```
public abstract void listen() throws ConfirmServiceException
```

可以使用此方法监听确认消息。收到确认消息时，通过使用消息中的信息可以创建 `ConfirmResponse` 对象并调用 `messageReceived` 方法。

ConfirmResponse 对象需要下表中显示的信息。

表 9-1 ConfirmResponse 参数

参数	描述
pushType	收到的消息的类型。使用在 PushConstants 类中定义的类型之一（请参见第 8-6 页上的第 8.12 节 “PushConstants 类”）。
messageID	由外部实体指定以标识消息的 ID。对于来自 MMSC 的消息，此 ID 是由 MMSC 指定的。例如，如果使用 UDP 并且消息包含在 responseObject 中，则该值可能为 Null。
responseStatus	响应的状态。该值可能为 Null。
responseDescription	响应的描述。该值可能为 Null。
responseObject	包含确认消息的二进制对象。例如，如果消息来自 MMSC 并且由消息 ID 进行标识，则该对象可能为 Null。

9.2.4 messageReceived 方法

```
protected void messageReceived(ConfirmResponse confirmResponse)
throws ConfirmServiceException
```

可以使用此方法将确认消息中收到的信息发送到 Content Delivery Server。从 listen 方法的实现中调用此消息。

9.3 使用确认服务 API

cdsapi.jar 文件中提供了确认服务 API 的类。在编译适配器时，此文件必须位于类路径中。为方便起见，$CDS_HOME/dist/cds/staging/jar 目录中提供了所有 Content Delivery Server JAR 文件的副本。在编译所创建的适配器时，请在类路径中使用此缓冲区。

使适配器可用于 Content Delivery Server 取决于所使用的应用服务器，以及是否已部署了该服务器。要使适配器可用，请执行以下操作：

1. **为适配器创建 JAR 文件。**
2. **对于所有应用服务器，请将该 JAR 文件放在 $CDS_HOME/dist/cds/lib/external 目录中。**

   现在，该适配器将包含在所有以后的部署中。

3. **如果现有部署需要使用该适配器，请将该 JAR 文件放在每个部署的** `$CDS_HOME/deployment/`*deployment-name*`/lib/external` **目录中。**

   如果使用的是 WebLogic Server，则会为您处理类路径。

   如果使用的是 Sun Java System Application Server，则会为每个部署更新类路径：

   a. **在编辑** `$CDS_HOME/deployment/`*deployment-name*`/sun/domains/cdsdomain/config/domain.xml` **文件之前，先对其进行备份，以便从编辑期间可能引入的任何错误中恢复。**

   b. **编辑** `domain.xml` **并修改** `java-config` **元素，将 JAR 文件的绝对路径添加到** `classpath-suffix` **属性中。**

   c. **保存所做的更改。**

4. **编辑** `$CDS_HOME/deployment/`*deployment-name*`/conf` **目录中的** `ConfirmListener.properties` **文件。**

   a. **定义另一个确认服务处理程序。**

   以下代码示例显示了要添加到 `ConfirmListener.properties` 中的语句。

```
confirmservice.handler=handler-id
confirmservice.handler.handler-id.classname=adapter-name
confirmservice.handler.handler-id.processor.classname=confirm-processor-name
confirmservice.handler.handler-id.pushtype=push-type
```

   handler-id 是任意唯一字符串，用于标识所定义的处理程序；*adapter-name* 是 `ConfirmServiceAdapter` 类实现的全限定类名。 *confirm-processor-name* 为以下值之一：

   - `com.sun.content.server.confirmservice.mms.MMSConfirmProcessor` - 如果确认服务适配器与 MMSC 配合使用，则使用此值。
   - `com.sun.content.server.confirmservice.udp.UDPConfirmProcessor` - 如果确认服务适配器与 UDP 配合使用，则使用此值。

   push-type 是以下值之一：

   - `mms` - 如果确认服务适配器与 MMSC 配合使用，则使用此值。
   - `udp` - 如果确认服务适配器与 UDP 配合使用，则使用此值。

   b. **保存所做的更改。**

5. **重新启动任何现有部署以识别新的 JAR 文件。**

第 10 章

流式传输 API

Sun Java System Content Delivery Server 流式传输 API 在 Content Delivery Server 和所选的流式传输服务器之间提供一个接口。可以使用此 API 编写流式传输适配器以使用流式传输服务器。

流式传输 API 包含以下类和接口：

- `StreamingAdapter` - 可以实现与流式传输服务器进行通信的接口。
- `StreamingContent` - 包含内容相关信息的类。
- `StreamingException` - 在出现错误时流式传输 API 抛出的异常。
- `StreamingServerConnectionFailedException` - 在 Content Delivery Server 无法连接到流式传输服务器时流式传输 API 抛出的异常。
- `StreamingSubscriber` - 此类包含请求内容的订户的相关信息。

流式传输 API 的类和接口位于 `com.sun.content.server.streaming` 软件包中。有关这些类和接口的其他信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 中的 Javadoc 工具的 HTML 输出。

10.1 一般处理流程

流式传输适配器是 `StreamingAdapter` 接口的实现，用于管理将数据传输到流式传输服务器以及向订户传送数据的过程。

本节介绍了 Content Delivery Server 和流式传输服务器之间的以下交互：

- 储存
- 取消储存
- 购买

10.1.1 储存

当 Vending Manager 储存按需流式传输的内容时，Content Delivery Server 将创建一个描述内容的 `StreamingContent` 对象。然后，调用 `uploadContent` 方法以将内容传输到流式传输服务器。如果出现错误，流式传输适配器必将抛出 `StreamingException`。如果流式传输适配器无法连接到流式传输服务器，则流式传输适配器必将抛出 `StreamingServerConnectionFailedException`。当 Content Delivery Server 看到异常时，将停止储存过程，并且不储存内容。

10.1.2 取消储存

当 Vending Manager 取消储存按需流式传输的内容时， Content Delivery Server 将创建一个描述内容的 `StreamingContent` 对象，并调用 `deleteContent` 方法以便从流式传输服务器中删除该内容。如果出现错误，则流式传输适配器必将抛出 `StreamingException`。如果流式传输适配器无法连接到流式传输服务器，流式传输适配器必须抛出 `StreamingServerConnectionFailedException`。如果 Content Delivery Server 看到异常，仍会将内容标记为已取消储存，并且订户无法使用该内容。

10.1.3 购买

当订户购买内容或请求免费内容时，将执行以下操作之一：

- 如果内容项是按需流式传输的内容，并且流式传输服务器提供了通过 URL 访问内容的权限，则 Content Delivery Server 将使用标识该内容项的信息调用 `getStreamingURL`。此方法返回订户用于从流式传输服务器开始接收流式传输内容的 URL。
- 如果内容项是按需流式传输的内容，并且流式传输服务器提供了通过会话描述协议 (Session Descripto Protocol, SDP) 文件访问内容的权限， Content Delivery Server 将使用标识内容项的信息调用 `getSDP` 方法。此方法返回订户用于从流式传输服务器开始接收内容的 SDP 文件。
- 如果内容项是 SDP 文件表示的实时内容， Content Delivery Server 将调用 `addInfoToSDPContent` 方法，以便将编码的信息添加到 SDP 文件中。通常，这是特定于 Content Delivery Server 的信息，后者将更改 SDP 文件中包含的实时流式传输协议 (Real Time Streaming Protocol, RTSP) URL。此方法返回订户用于从流式传输服务器开始接收内容的已修改的 SDP 文件。

10.2 StreamingAdapter 接口

StreamingAdapter 接口用于与所使用的流式传输服务器进行通信。请实现此接口，以便为流式传输服务器创建流式传输适配器。

10.2.1 addInfoToSDPContent 方法

```
public byte[] addInfoToSDPContent(byte[] binary, long categoryItemId,
String contentId, String inCodedTicket, String fileExtension, String
contentName, long mimeTypeId, String vendingAccountId) throws
StreamingServerConnectionFailedException, StreamingException;
```

当订户购买 SDP 表示的流式传输内容时， Content Delivery Server 将调用 addInfoToSDPContent 方法。可以使用此方法，将其他编码的信息添加到 SDP 文件中。例如，可以使用此方法在 URL 中添加 Content Delivery Server 或所使用的流式传输服务器可能需要的其他参数，如编码证明书或订户标识符。如果出现错误，请确保实现抛出相应的异常。

10.2.2 deleteContent 方法

```
public void deleteContent(StreamingContent streamingContent, String
vendingAccountId) throws StreamingServerConnectionFailedException,
StreamingException;
```

当 Vending Manager 取消储存流式传输的内容时， Content Delivery Server 将调用 deleteContent 方法。可以使用此方法从流式传输服务器中删除内容。应传递描述内容的对象以及储存内容的 Vending Manager 的 ID。如果出现错误，请确保实现抛出相应的异常。

10.2.3 getSDP 方法

```
public byte[] getSDP(long categoryItemId, String contentId, String
inCodedTicket, String fileExtension, String contentName, String
vendingAccountId, String liveURL) throws
StreamingServerConnectionFailedException, StreamingException;
```

当订户请求访问 SDP 文件表示的流式传输内容时，将会调用 getSDP 方法。可以使用此方法为 contentId 标识的内容返回 SDP。如果流式传输服务器仅提供了通过 URL 访问内容的权限，则不需要实现此方法。如果出现错误，请确保实现抛出相应的异常。

10.2.4 getStreamingURL 方法

```
public String getStreamingURL(long categoryItemId, String contentId,
String inCodedTicket, String fileExtension, String contentName, long
mimeTypeId, String vendingAccountId, String liveURL) throws
StreamingServerConnectionFailedException, StreamingException;
```

当订户购买流式传输的内容时，将会调用 `getStreamingURL` 方法。 Content Delivery Server 提供了标识内容和购买事务的信息。可以使用此方法提供订户访问内容项所需的 URL。如果流式传输服务器未提供通过 URL 访问内容的权限，则不需要实现此方法。如果出现错误，请确保实现抛出相应的异常。

10.2.5 returnsSDP 方法

```
public boolean returnsSDP();
```

Content Delivery Server 调用 `returnsSDP` 方法以确定是通过 SDP 文件还是 URL 来访问流式传输服务器上的内容。如果通过 SDP 文件访问内容，则返回 `true` 并实现 `getSDP` 方法。如果通过 URL 访问内容，则返回 `false` 并实现 `getStreamingURL` 方法。

10.2.6 uploadContent 方法

```
public void uploadContent(StreamingContent streamingContent, String
vendingAccountId) throws StreamingServerConnectionFailedException,
StreamingException;
```

当 Vending Manager 储存流式传输的内容时， Content Delivery Server 将调用 `uploadContent` 方法。可以使用此方法将内容复制到流式传输服务器。应传递描述内容的对象以及储存内容的 Vending Manager 的 ID。如果出现错误，请确保实现抛出相应的异常。

10.3 使用流式传输 API

位于 `$CDS_HOME/deployment/`*deployment-name*`/lib/cdslib` 目录中的 `cdsapi.jar` 提供了流式传输 API 的类。在编译适配器时，此文件必须位于类路径中。为方便起见， `$CDS_HOME/dist/cds/staging/jar` 目录中提供了所有 Content Delivery Server JAR 文件的副本。在编译所创建的适配器时，请在类路径中使用此缓冲区。

使适配器可用于 Content Delivery Server 取决于所使用的应用服务器，以及是否已部署了该服务器。要使适配器可用，请执行以下操作：

1. **为适配器创建 JAR 文件。**
2. **对于所有应用服务器，请将该 JAR 文件放在** `$CDS_HOME/dist/cds/lib/external` **目录中。**

   现在，该适配器将包含在所有以后的部署中。
3. **如果现有部署需要使用该适配器，请将该 JAR 文件放在每个部署的** `$CDS_HOME/deployment/`*deployment-name*`/lib/external` **目录中。**

   如果使用的是 WebLogic Server，则会为您处理类路径。

   如果使用的是 Sun Java System Application Server，则会为每个部署更新类路径：

   a. **在编辑** `$CDS_HOME/deployment/`*deployment-name*`/sun/domains/cdsdomain/config/domain.xml` **文件之前，先对其进行备份，以便从可能在编辑期间引入的任何错误中恢复。**

   b. **编辑** `domain.xml` **并修改** `java-config` **元素，将 JAR 文件的绝对路径添加到** `classpath-suffix` **属性中。**

   c. **保存所做的更改。**
4. **在** `$CDS_HOME/deployment/`*deployment-name*`/conf` **目录下的** `streaming.properties` **文件中，将** `streaming.adapter.impl.class` **属性设置为流式传输适配器的全限定名称。**
5. **重新启动任何现有部署以识别新的 JAR 文件。**

第 11 章

订户 API

通过订户 API，可以访问由 Content Delivery Server 维护的数据。使用此 API 可以获取必要的数据来创建自己的客户机应用程序，以便订户访问由 Content Delivery Server 管理的内容。

位于具备 Content Delivery Server Subscriber Portal 组件的服务器上以 Java 编程语言编写的客户机可以直接调用订户 API。如果编写客户机的编程语言不是 Java，或者客户机所在的服务器不包含 Subscriber Portal，则客户机必须通过 XML-RPC（Remote Procedure Call，远程过程调用）实现来访问订户 API。

订户 API 包括客户机可能使用的以下类和接口：

- `ApiContextFactory` - 此类创建包含订户特征（如语言环境、设备型号和移动 ID）的 `IApiContext` 对象。
- `ApiServiceFactory` - 此类为特定订户创建服务。通过服务可以访问相关信息的集合。例如，通过内容服务可以访问订户上次购物的列表、各个项目的详细信息和订户书签内容。
- `ApiUtil` - 此类提供多种功能，例如启动事务或检查数据库版本。
- `IApiContext` - 此接口提供当前订户的相关信息。
- `ICategoryService` - 此接口提供对类别树和订户类别列表的访问。
- `IContentService` - 通过此接口可以访问内容以进行浏览、查找、检索和购买，还可以访问预览文件。
- `IDownloadService` - 此接口提供对内容描述符的访问。
- `IGiftingService` - 订户可通过此接口将礼品或有关某一项的消息发送给其他订户。
- `IMessageService` - 通过此接口可以将消息发送给此订户或其他订户。消息可以是电子邮件、 MMS、 SMS 或 WAP 推送格式。
- `IStreamingService` - Content Delivery Server 可通过此接口访问流式传输的内容，流式传输服务器可通过此接口验证内容请求。
- `ISystemService` - 此接口提供对系统级内容（如语言环境、内容类型和设备型号）的访问。
- `IUserService` - 此接口提供对订户相关信息的访问，并可用来创建新的订户帐户。

- `CDSException` - 出现错误时订户 API 抛出的异常。

有关这些类以及其他类的其他信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 中的 Javadoc 工具的 HTML 输出。

11.1 一般处理流程

本节介绍可能需要客户机应用程序执行的常规任务。本节中使用的类名是指构成订户 API 的类。如果您使用 XML-RPC 访问 Content Delivery Server 数据，请参见第 11-6 页上的第 11.3 节 “XML-RPC 实现”以获得等效的处理程序信息。

可以使用验证密钥来验证对 Content Delivery Server 的访问请求。该密钥包含在 `IApiContext` 对象中，它是在创建 `IApiContext` 对象时生成的。需要使用验证密钥才能创建服务；如果在指定时间 （几分钟）内未使用该密钥，它将会失效。默认时间为 10 分钟。要更改此时间，请在 `$CDS_HOME/deployment/`*`deployment-name`*`/conf/SubscriberPortal.properties` 文件中设置 `subscriberApi.authkey.timeout.minutes` 属性。如果密钥失效，则必须创建新的 `IApiContext` 对象以获得新密钥。

通常，您为订户接口创建的客户机应用程序包含多个操作，如下面列表中所述。

- 通过调用 `ApiContextFactory` 来创建 `IApiContext`。

  `IApiContext` 对象说明特定订户或匿名订户。所有服务都使用此对象检索特定于对象所说明的订户的数据。通常，您应该为每个用户会话创建一个 `IApiContext` 对象，并将验证密钥存储在 `HttpSession` 对象中。要获得验证密钥，请调用 `IApiContext.getAuthKey` 方法。有关样例代码，请参见第 11-4 页上的第 11.2.2 节 “IApiContext 对象创建示例”。

- 通过调用 `ApiServiceFactory` 来创建获得要使用的信息所需的服务。

  创建的服务取决于要执行的任务。例如，要使某位订户能够为另一订户购买礼品，请创建 `IGiftingService` 对象。要为订户提供已购买内容的列表，请创建 `IContentService` 对象。您必须具有验证密钥才能创建服务。创建的每个服务都提供特定于 `IApiContext` 对象所说明的订户的数据。有关样例代码，请参见第 11-5 页上的第 11.2.3 节 “创建服务的示例”。

- 通过为所创建的服务调用方法来检索要使用的信息。

  例如，如果创建了 `IContentService`，则可以调用 `getCampaigns` 方法以获得订户可用的活动列表，或调用 `getPurchases` 方法以获得订户已购买的项列表。

- 注销 Content Delivery Server。

  在完成后，可通过调用 `IApiContextFactory.removeApiContext` 方法删除验证密钥。

11.2 使用订户 API

订户 API 的类位于软件包 `com.sun.content.server.subscriberapi` 中。此软件包包含在以下某个位置的 `subscriberportal.jar` 文件中：

- 对于 Sun Java System Application Server，该文件位于 `$CDS_HOME/deployment/`*deployment-name*`/sun/domains/`*server-domain*`/applications/j2ee-modules/CDSSubscriberPortal/WEB-INF/lib`。
- 对于 WebLogic Server，该文件位于 `$CDS_HOME/deployment/`*deployment-name*`/weblogic/domains/`*server-domain*`/applications/subscriber/WEB-INF/lib`。

deployment-name 是在部署 Catalog Manager 时指定的名称；*server-domain* 是在部署配置文件中为 `app.server.domain` 属性指定的值。

如果您的客户机应用程序是 Java 应用程序，则请使用 `subscriberportal.jar` 文件中的订户 API 类创建您的客户机。编译应用程序时，此 JAR 文件必须位于类路径中。

要执行应用程序，请将包含客户机的 JAR 文件置于 `$CDS_HOME/deployment/`*deployment-name*`/.../lib` 目录中（该目录还包含 `subscriberportal.jar` 文件）。客户机必须运行在与随 Content Delivery Server 一起提供的 Subscriber Portal 相同的 Web 应用程序结构中。不支持独立的 Java 应用程序。

如果客户机不是 Java 应用程序或者与 Subscriber Portal 不在同一台服务器上，请参见第 11-6 页上的第 11.3 节“XML-RPC 实现”获取有关通过 XML-RPC 访问订户 API 的信息。

11.2.1 管理事务

每次创建服务实例并调用其方法来执行任务时，都会有事务发生。应用程序必须按照如下步骤中说明的方式管理 Content Delivery Server 事务：

1. **调用服务之前，请调用 `ApiUtil.initTransaction` 方法指明事务的起点。**
2. **如果事务成功完成，请调用 `ApiUtil.commitTransaction` 方法提交完成的工作。如果处理事务的过程中出错，请调用 `ApiUtil.rollbackTransaction` 方法终止工作并将数据恢复到以前的状态。**
3. **事务资源必须在调用 `ApiUtil.disposeTransaction` 方法的 `finally` 块中释放。**

有关样例实现，请参见第 11-4 页上的第 11.2.2 节 “IApiContext 对象创建示例”和第 11-5 页上的第 11.2.3 节 “创建服务的示例”中的代码示例。

11.2.2 IApiContext 对象创建示例

以下节选代码显示如何创建 IApiContext 对象。

编码样例 11-1 使用 Java 类文件来创建 IApiContext 对象

```
...
try
{
  // Open a Transaction
  ApiUtil.initTransaction();

  // Create a map of credentials (from user input)
  Properties credentials = new Properties();
  credentials.put(ApiContextFactory.CREDENTIAL_USERNAME, username);
  credentials.put(ApiContextFactory.CREDENTIAL_PASSWORD, password);

  // Attempt to authenticate using the credentials
  IApiContext apiContext = ApiContextFactory.createApiContext(credentials);

  // Save the IApiContext authentication key in the HttpSession
  session.setAttribute("API_CONTEXT_AUTH_KEY", apiContext.getAuthKey());

  // Commit the Transaction
  ApiUtil.commitTransaction();
}
catch (CDSException e)
{
  // Rollback the Transaction
  ApiUtil.rollbackTransaction();

  // Evaluate the exception's error code
  if(e.getErrorCode().equals(CDSException.CDS_EX_SUBSCRIBER_DISABLED)
  {
    // handle disabled user
    ...
  }
  else
  {
    // handle API Exception
    ...
  }
}
finally
{
  // clean up Transaction
  ApiUtil.disposeTransaction();
}
...
```

11.2.3 创建服务的示例

以下节选代码显示如何创建内容服务以及如何使用该服务购买内容。

编码样例 11-2 创建服务

```
...
try
{
  // Open a Transaction
  ApiUtil.initTransaction();

  // Retrieve the IApiContext authentication key from the HttpSession
  String apiContextAuthKey =
      (String)session.getAttribute("API_CONTEXT_AUTH_KEY");

  // Get a reference to a Content Service
  IContentService cs = ApiServiceFactory.getContentService(apiContextAuthKey);

  // Attempt to purchase a content item as part of a campaign
  cs.purchaseContent(contentId, campaignId, true);

  // Commit the Transaction
  ApiUtil.commitTransaction();
}
catch (CDSException e)
{
  // Rollback the Transaction
  ApiUtil.rollbackTransaction();

  // Handle API Exception
  ...
}
finally
{
  // Clean up Transaction
  ApiUtil.disposeTransaction();
}
...
```

11.3 XML-RPC 实现

如果客户机不是 Java 应用程序，或者没有运行在具有 Content Delivery Server 的服务器上，则客户机必须使用 XML-RPC 与 Content Delivery Server 进行通信。通过 XML-RPC，客户机可以在进行传输和数据编码时分别使用 HTTP 和 XML 进行远程过程调用。通过使用 Internet 上的绑定，可以将 XML-RPC 与许多不同编程语言一起使用。订户 API 的所有这些功能通过 XML-RPC 才可以使用。

注 – 有关 XML-RPC 的教程不在本文档的范围之内。您可以从 Internet 的多个不同 Web 站点中获取有关编写使用 XML-RPC 的应用程序方面的信息。

11.3.1 访问 Content Delivery Server

要从 Content Delivery Server 获取数据，客户机必须能够与 Content Delivery Server 进行通信。请要求网络管理员确保客户机可以访问 Content Delivery Server，并且任何所需的代理或防火墙均已配置为允许该访问。

此外， Content Delivery Server 还必须认可客户机已授权执行数据请求。`$CDS_HOME/deployment/`*`deployment-name`*`/conf/SubscriberPortal.properties` 文件中的 `subscriberApi.xml-rpc.trustedHosts` 属性包含请求将被接受的主机列表。

将 `subscriberApi.xml-rpc.trustedHosts` 属性设置为客户机所在主机的主机名和 IP 地址，而不管是否与 Content Delivery Server 位于同一主机。要接受任何主机的请求，请保留此值为空。要接受多个主机的请求，请使用逗号分隔主机名或 IP 地址，例如：

```
subscriberApi.xml-rpc.trustedHosts=127.0.0.1,localhost
```

11.3.2 使用订户 API 的 XML-RPC 处理程序

不论是直接访问订户 API 还是通过 XML-RPC 访问，应用程序的一般流程（请参见第 11-2 页上的第 11.1 节“一般处理流程”）是类似的。在这两种情况下，需要具有 API 上下文对象才能访问服务。在直接访问订户 API 时，将使用验证密钥来创建服务。在使用 XML-RPC 时，可以使用 `AuthenticationHandler` 获取将传递到每种方法的 API 上下文对象。

XML-RPC 实现中的处理程序执行与订户 API 服务相同的功能。每个处理程序及其相应的服务都具备带有等效参数的等效方法。

本节介绍以下主题：

11.3.2.1 XML-RPC 方法调用的指导

请按以下指导原则编写处理程序调用代码。样例代码是使用 Java 编程语言编写的。

- 将调用的方法参数置于一个散列表中。将散列表置于向量中。

```
...
// Set up the input parameters
Vector parameters = new Vector();
Hashtable ht = new Hashtable();
ht.put("username", "user1");
ht.put("password", "cryptic1");
parameters.addElement(ht);
...
```

- 要调用方法，请传递方法的名称，以及为包含参数的散列表创建的向量。将返回一个散列表。方法调用必须包含处理程序的名称，例如：`AuthenticationHandler.getApiContext`。

```
...
// Send the request to Content Delivery Server
Hashtable response =
(Hashtable) client.execute("AuthenticationHandler.getApiContext", parameters);
...
```

- 通过检查返回的散列表中包含的响应代码，验证方法是否成功执行。如果发生错误，散列表中还包含响应消息。

```
...
// Evaluate the response
String errorCode = (String)response.get("response_code");
```

```
if (!errorCode.equals("1"))
{
  // Handle Error
  System.out.println((String)response.get("response_message"));
}
```

- 如果方法已成功执行，则提取散列表中返回的值。如果方法未返回任何值，则散列表只包含响应代码。

```
...
  // Authentication successful
  Hashtable apiContext = (Hashtable)response.get("apiContext");
  Integer subscriberId = (Integer)apiContext.get("subscriberId");
  String username = (String)apiContext.get("username");
  String localeCode = (String)apiContext.get("localeCode");
  String mobileId = (String)apiContext.get("mobileId");
  Integer modelId = (Integer)apiContext.get("modelId");
  ...
```

以下几节介绍可以使用的处理程序。第 11-38 页上的第 11.3.3 节“使用处理程序的示例”中提供示例。

11.3.2.2 AuthenticationHandler

AuthenticationHandler 等效于 ApiContextFactory 类。该处理程序创建包含订户特征的 API 上下文对象。API 上下文对象等效于 IApiContext 对象。调用处理程序中的方法的指导在第 11-7 页上的第 11.3.2.1 节 “XML-RPC 方法调用的指导”中进行了介绍。要调用方法，请将处理程序的名称与方法的名称相连接，例如：

```
AuthenticationHandler.getApiContext
```

下表介绍 AuthenticationHandler 的方法。

表 11-1 AuthenticationHandler 的方法

方法名称	描述	参数	返回*
getAnonymousApiContext	为匿名订户创建 API 上下文对象。	localeCode, modelId	apiContext
getApiContext	根据提供的信息验证订户，并创建包含该订户信息的 API 上下文对象。	下列项目之一： • username 和 password • mobileId • uniqueId • subscriberId • requestHeaders	apiContext

* 除了列出的对象，所有方法均返回 response_code；如果发生错误则返回 response_message。

11.3.2.3 CategoryHandler

CategoryHandler 等效于 ICategoryService 接口。该处理程序提供对类别树和订户类别列表的访问。调用处理程序中的方法的指导在第 11-7 页上的第 11.3.2.1 节 “XML-RPC 方法调用的指导”中进行了介绍。要调用方法，请将处理程序的名称与方法的名称相连接，例如：

```
CategoryHandler.getCategory
```

下表介绍 CategoryHandler 的方法。

表 11-2 CategoryHandler 的方法

方法名称	描述	参数	返回\
getCategory	获取当前订户的指定类别。	apiContext, categoryId, includeContentCount	category
getCategoryBranch	从 categoryId 中指定的类别开始，获取整个类别树分支。要获取整个树，请指定根类别。返回的类别包含 contentTypeIdList 中列出的内容类型以及 statusList 中列出的状态。如果未提供 contentTypeIdList，则考虑所有内容类型。如果未提供 statusList，则考虑所有状态。	apiContext、categoryId、contentTypeIdList（可选）、statusList（可选）、includeContentCount	category
getNotEmptySubCategories	对于 categoryId 中指定的类别，获取子类别列表，这些子类别包含 contentTypeIdList 中列出的内容类型，及 statusList 中列出的状态。如果未提供 contentTypeIdList，则考虑所有内容类型。如果未提供 statusList，则考虑所有状态。	apiContext、categoryId、contentTypeIdList（可选）、statusList（可选）、includeContentCount	categoryList
getSubCategories	对于 categoryId 中指定的类别，获取所有子类别的列表。	apiContext, categoryId, includeContentCount	categoryList
getRootCategory	获取当前订户的根类别。	apiContext, includeContentCount	category

表 11-2 CategoryHandler 的方法 （续）

方法名称	描述	参数	返回\
hideCategory*	隐藏 categoryIds 中指定的、订户选择不查看的类别，并返回更新后的列表。	apiContext, categoryIds, categoryList	categoryList
moveCategoryDown*	在下一活动类别下方将 categoryIds 中指定的每个类别向下移动一个位置，并返回更新后的列表。	apiContext, categoryIds, categoryList	categoryList
moveCategoryUp*	在下一活动类别上方将 categoryIds 中指定的每个类别向上移动一个位置，并返回更新后的列表。	apiContext, categoryIds, categoryList	categoryList
showCategory*	显示 categoryIds 中指定的、订户已选择的类别，并返回更新后的列表。	apiContext, categoryIds, categoryList	categoryList
updateCategories	为 categoryList 中的类别更新数据库中的信息。	apiContext, categoryList	无

* 该方法更改内存中保留的类别列表。要永久更改，必须调用 updateCategories。

\ 除了列出的对象，所有方法均返回 response_code；如果发生错误则返回 response_message。

11.3.2.4 ContentHandler

ContentHandler 等效于 IContentService 接口。该处理程序提供对内容的访问，以便完成浏览、查找、检索和购买。调用处理程序中的方法的指导在第 11-7 页上的第 11.3.2.1 节 “XML-RPC 方法调用的指导”中进行了介绍。要调用方法，请将处理程序的名称与方法的名称相连接，例如：

```
ContentHandler.browseContent
```

下表介绍 ContentHandler 的方法。

表 11-3 ContentHandler 的方法

方法名称	描述	参数	返回*
addBookmark	将一项内容添加到订户的书签内容列表。	apiContext, contentId	无
browseContent	获取 categoryId 指定类别中具有 contentTypeIdList 指定类型的 numberToReturn 指定数量的项目。如果未提供 contentTypeIdList，则包含所有内容类型。 **注 –** 此方法已过时。请使用 searchContent 方法的新版本。	apiContext、categoryId、contentTypeIdList（可选）、startIndex、numberToReturn	contentList, totalSize
cancelSubscription	取消订户对某项内容的订阅。	apiContext, contentId	无
clearBookmarks	清除订户的书签内容列表。	apiContext	无
deleteBookmark	从订户的书签内容列表中删除某项内容。	apiContext, contentId	无
getAnonymousCampaignForCoupon	获取与指定礼券相关的活动信息。如果订户是匿名的，请调用此方法。	apiContext, couponCode	campaign
getBookmarks	获取订户书签中已有项目列表。	apiContext	contentList
getBundledItems	获取包中的内容项。	apiContext, contentId, criteria	contentList, totalSize
getCampaign	获取活动信息。	apiContext, campaignId	campaign
getCampaignForCoupon	获取与指定礼券相关的活动信息。如果已知订户，请调用此方法。	apiContext, couponCode	campaign
getCampaignItems	获取与活动相关的内容列表。	apiContext, campaignId, startIndex, numberToReturn	contentList, totalSize
getCampaigns	获取订户可用的活动列表。	apiContext	campaignList

表 11-3 ContentHandler 的方法 （续）

方法名称	描述	参数	返回*
getContentByClassId	获取由 contentClassId 标识的内容的特定版的内容 ID。	apiContext, contentClassId	contentId
getContentByKeyword	获取由 contentKeyword 标识的内容的特定版的内容 ID。	apiContext, contentKeyword	contentId
getContentDetails	获取某项内容的信息。	apiContext, contentId, criteria	content
getContentDetailsCriteria	获取一个空条件对象，它可用于接受条件作为参数的方法。	apiContext	criteria
getContentDetailsList	获取 contentIdList 中指定的每项内容的信息。	apiContext, contentId, criteria	contentList
getContentSummary	获取某项内容的概要信息。	apiContext, contentId	contentSummary
getDeliveryType	获取用于传送此内容的传送类型。	apiContext, contentId	deliveryType
getPurchasedBundles	获取订户购买的包含 contentId 中指定内容项的包列表。	apiContext, contentId	contentList, totalSize
getPurchases	获取订户已购买内容列表。	apiContext	purchaseList
getSupportedModels	获取可运行内容的型号列表。	apiContext, contentId	modelList
getTicket	获取由 contentId 标识的内容的购买证明书。	apiContext、contentId、bundleId（仅当内容是包的一部分才指定）	codedTicket
hasPurchases	确定订户是否购买任何内容。	apiContext	hasPurchases
isBookmarked	确定订户是否为内容编制书签。	apiContext, contentId	isBookmarked
isContentInCampaign	确定 contentId 指定的内容是否位于 campaignId 指定的活动中。	apiContext, contentId, campaignId	isContentInCampaign
isMMSCapable	确定内容是否可以使用 MMS 推送到设备。	apiContext, contentId	isMMSCapable

表 11-3 ContentHandler 的方法 （续）

方法名称	描述	参数	返回*
isSMSCapable	确定是否可以使用 SMS 将内容推送到设备。	apiContext, contentId	isSMSCapable
purchaseContent	购买内容。如果指定了 campaignID，则购买属于活动一部分的内容。匿名订户无法使用。	apiContext、contentId、campaignId （可选）、 isSkipTrial	无
requestContent	将内容发送给某个订户。必须先购买内容。	apiContext, contentId, requestParams, maxNumberToSend	wasDelivered
searchContent	获取与查找查询匹配的内容的内容概要列表。	apiContext, searchQuery, sortKeyMap, categoryId, startIndex, numberToReturn	contentList, totalSize
searchContent	获取匹配查找条件的内容列表。从 categoryId 中指定的类别开始查找类别树。要查找所有内容，请指定根类别 （请参见 CategoryHandler 中的 getRootCategory 方法）。 **注 –** 此方法已过时。请使用新版本。	apiContext, categoryId, contentTypeIdList （可选）、 keyword、startIndex、numberToReturn	contentList, totalSize

* 除了列出的对象，所有方法均返回 response_code；如果发生错误则返回 response_message。

11.3.2.5 DownloadHandler

DownloadHandler 等效于 IDownloadService 接口。该处理程序提供对下载内容所需描述符的访问。调用处理程序中的方法的指导在第 11-7 页上的第 11.3.2.1 节 “XML-RPC 方法调用的指导”中进行了介绍。要调用方法，请将处理程序的名称与方法的名称相连接，例如：

```
DownloadHandler.downloadConfirm
```

下表介绍 DownloadHandler 的方法。

表 11-4 DownloadHandler 的方法

方法名称	描述	参数	返回*
downloadConfirm	确认内容已下载到设备。	apiContext, codedTicket, isOneStepConfirm, status	无
downloadContent	为 contentId 指定的内容项下载二进制文件。	apiContext, contentId	contentLength, contentType, descriptorData
downloadContentDescriptor	创建内容描述符文件，并将其返回给调用者。	apiContext, codedTicket	contentLength, contentType, descriptorData
downloadDelete	确认从设备中删除了该内容。	apiContext, codedTicket, isOneStepConfirm, status	无
downloadJAD	创建 Java 应用程序描述符 (Java Application Descriptor，JAD) 文件，并将其返回给调用者。	apiContext, codedTicket	contentLength, contentType, descriptorData
downloadJAM	为 iAppli 应用程序创建应用程序描述符文件，并将其返回给调用者。	apiContext, codedTicket	contentLength, contentType, descriptorData
pushMMSContent	使用 MMS 将内容推送到设备。 **注** – 此方法已过时。请使用 pushMMSContentByTicket 方法。	apiContext, contentId	无
pushMMSContentByTicket	使用 MMS 将 codedTicket 所标识的内容推送到设备。	apiContext, codedTicket	无
pushSMSContentBinary	使用 SMS 将外部内容推送到设备。	apiContext, mobileId, contentBinary, mimeType, contentType, smsParams	无
pushSMSContentByTicket	使用 SMS 将 codedTicket 所标识的内容推送到设备。	apiContext, codedTicket, smsParams	无
sendInstall	将内容安装文件推送到订户设备。	apiContext, codedTicket	无

* 除了列出的对象，所有方法均返回 response_code；如果发生错误则返回 response_message。

11.3.2.6 GiftingHandler

GiftingHandler 等效于 IGiftingService 接口。该处理程序将有关内容的礼品或通知发送到其他订户。调用处理程序中的方法的指导在第 11-7 页上的第 11.3.2.1 节“XML-RPC 方法调用的指导”中进行了介绍。要调用方法，请将处理程序的名称与方法的名称相连接，例如：

```
GiftingHandler.createGifting
```

下表介绍 GiftingHandler 的方法。

表 11-5 GiftingHandler 的方法

方法名称	描述	参数	返回*
cancelGifting	取消礼品订阅。	apiContext, giftId	无
checkAndExpireGifting	确定礼品是否已过期。	apiContext, gifting	isGiftExpired
getGiftingById	获取 giftId 指定的有关礼品的信息。	apiContext, giftId, filter, bundledContentId	gifting
getGiftingByTicket	获取 giftTicket 指定的有关礼品的信息。	apiContext, giftTicket, filter, bundledContentId	gifting
getGiftingsByGifter	获取订户给出的所有礼品的信息。	apiContext, filter	giftingList
getGiftingsByRecipient	获取订户接收到的所有礼品的信息。	apiContext, filter	giftingList
giftContent	购买某项内容作为给其他订户的礼品。	apiContext、contentId、campaignId（可选）、recipientApiContext、message、giftedDownloads、giftedSubscriptions	giftId
isGiftingUsed	确定收件人是否需要礼品所提供的所有用途。	apiContext, gifting	isGiftingUsed
messageContent	发送有关某项内容的消息给其他订户。	apiContext, contentId, recipientApiContext, message	giftId

* 除了列出的对象，所有方法均返回 response_code；如果发生错误则返回 response_message。

11.3.2.7 MessageHandler

MessageHandler 等效于 IMessageService 接口。该处理程序将电子邮件、SMS、WAP 推送以及 MMS 消息发送到当前订户或其他订户。调用处理程序中的方法的指导在第 11-7 页上的第 11.3.2.1 节 “XML-RPC 方法调用的指导”中进行了介绍。要调用方法，请将处理程序的名称与方法的名称相连接，例如：

```
MessageHandler.sendMessageToSelf
```

下表介绍 MessageHandler 的方法。

表 11-6 MessageHandler 的方法

方法名称	描述	参数	返回*
sendMessageToMobileId	将消息发送到提供的移动 ID 所指定的号码。	apiContext, subject, message, url, mobileId, contentId, messageCategory	无
sendMessageToSelf	将消息发送到当前订户。	apiContext, messageType, subject, message, url, contentId, messageCategory	无
sendMessageToSubscriberId	将消息发送到提供的订户 ID 所标识的订户。	apiContext, messageType, subject, message, url, subscriberId, contentId, messageCategory	无
sendMessageToUsername	将消息发送到提供的用户名所标识的订户。	apiContext, messageType, subject, message, url, username, contentId, messageCategory	无
sendMMSContent	使用 MMS 推送将消息发送到当前订户。	apiContext, mobileId, contentBinary, name, contentType, mimeType, subject, message, modelId, contentId, messageCategory	无

* 除了列出的对象，所有方法均返回 response_code；如果发生错误则返回 response_message。

11.3.2.8 StreamingHandler

StreamingHandler 等效于 IStreamingService 接口。该处理程序为 Content Delivery Server 提供了有关流式传输会话的信息，流式传输服务器可通过该处理程序验证内容请求。调用处理程序中的方法的指导在第 11-7 页上的第 11.3.2.1 节 “XML-RPC 方法调用的指导”中进行了介绍。要调用方法，请将处理程序的名称与方法的名称相连接，例如：

```
StreamingHandler.authenticate
```

下表介绍 StreamingHandler 的方法。

表 11-7 StreamingHandler 的方法

方法名称	描述	参数	返回*
authenticate	验证 Content Delivery Server 是否已知请求流式传输内容的订户。	apiContext	isAuthenticated
authorize	验证请求流式传输内容的订户是否购买了所请求的项。	apiContext, codedTicket	isAuthenticated
endSession	通知 Content Delivery Server 流式传输会话已结束。	apiContext, codedTicket	无
errorInSession	通知 Content Delivery Server 在流式传输会话期间出现错误。	apiContext, codedTicket	无
startSession	通知 Content Delivery Server 流式传输会话已开始。	apiContext, codedTicket	无

* 除了列出的对象，所有方法均返回 response_code；如果发生错误则返回 response_message。

11.3.2.9 SystemHandler

SystemHandler 等效于 ISystemService 接口。该处理程序提供对系统级数据的访问，例如内容类型，语言环境和型号。调用处理程序中的方法的指导在第 11-7 页上的第 11.3.2.1 节 “XML-RPC 方法调用的指导”中进行了介绍。要调用方法，请将处理程序的名称与方法的名称相连接，例如：

```
SystemHandler.getContentTypes
```

下表介绍 SystemHandler 的方法。

表 11-8 SystemHandler 的方法

方法名称	描述	参数	返回*
createTicket	为订户创建证明书，用来检索某项内容。	apiContext, contentId	ticket
getContentTypes	获取 Content Delivery Server 中定义的所有内容类型的列表。	apiContext	contentTypeList
getCountries	获取 Content Delivery Server 中包含的所有国家/地区的列表。	apiContext	countryList
getCountry	获取单个国家/地区的信息。	apiContext, countryCode	country
getDefaultLocale	获取系统的默认语言环境。	apiContext	locale
getDefaultModel	获取默认设备模型。	apiContext	modelId
getLocale	获取单个语言环境的信息。	apiContext, localeCode	locale
getLocales	获取 Content Delivery Server 中包含的所有语言环境的列表。	apiContext	localeList
getManufacturers	获取 Content Delivery Server 中包含的所有设备生产商的名称。	apiContext	manufacturerList
getModel	获取设备的信息。	apiContext, modelId	model
getModelId	获取与 modelId 指定的用户代理相关的设备内部 ID。	apiContext, userAgent	modelId
getModels	获取 Content Delivery Server 支持的所有设备的列表。	apiContext	modelList
getModelsForManufacturer	获取 Content Delivery Server 中给定生产商的型号。仅返回未被隔离的型号。	apiContext, manufacturer	modelList
isPushEnabled	确定订户的设备是否是可推送的。	apiContext, modelId	isPushEnabled

表 11-8 SystemHandler 的方法 （续）

方法名称	描述	参数	返回*
isTicketValid	确定订户是否可以使用某项内容的证明书。	apiContext, ticket, contentId	isTicketValid
sendEvent	发送 eventType 中指定的系统事件。 **注** – 此方法已过时。不再需要使用此功能。	apiContext、subscriberId、mobileId、contentId （可选）、 eventType	无
sendEventWithParameters	使用一组事件参数来发送系统事件。	apiContext, requestParams	无

* 除了列出的对象，所有方法均返回 response_code；如果发生错误则返回 response_message。

11.3.2.10 UserHandler

UserHandler 等效于 IUserService 接口。通过此处理程序可以访问订户的有关信息以及创建新的订户帐户。调用处理程序中的方法的指导在第 11-7 页上的第 11.3.2.1 节 “XML-RPC 方法调用的指导”中进行了介绍。要调用方法，请将处理程序的名称与方法的名称相连接，例如：

```
UserHandler.getSubscriberId
```

下表介绍 UserHandler 的方法。

表 11-9 UserHandler 的方法

方法名称	描述	参数	返回*
disableSubscriberBySubscriberId	禁用订户 ID 所标识订户的帐户。	apiContext, subscriberId	无
disableSubscriberByUsername	禁用用户名所标识订户的帐户。	apiContext, username	无
getSubscriberId	获取用户名所标识订户的订户 ID。	apiContext, username	subscriberId
getUserPreferences	获取订户设置的首选项。	apiContext	preferenceList
getUserProperties	获取当前订户的信息。	apiContext	propertyList
getUserPropertiesBySubscriberId	获取订户 ID 所标识订户的信息。	apiContext, subscriberId	propertyList
getUserPropertiesByUsername	获取用户名所标识订户的信息。	apiContext, username	propertyList

表 11-9 UserHandler 的方法（续）

方法名称	描述	参数	返回*
provision	用外部订户数据库的信息填充 Content Delivery Server 中的订户帐户。	apiContext, uniqueId, modelId, mobileId, localeCode	subscriberId
resetPasswordBySubscriberId	将订户 ID 所标识订户的密码设置为系统生成的值。	apiContext, subscriberId, passwordRequiresReset	password
resetPasswordByUsername	将用户名所标识订户的密码设置为系统生成的值。	apiContext, username, passwordRequiresReset	password
setLocaleCode	更改订户的语言环境代码。	apiContext, localeCode	无
setModelId	将型号 ID 更改为匹配订户使用的新设备。	apiContext, modelId	无
setPassword	更改订户的密码。	apiContext, password	无
setUserPreferences	设置订户选择的首选项。这些首选项管理显示给用户的信息。	apiContext, preferenceList	无
setUserProperties	设置订户信息。	apiContext, propertyList	无
signup	在 Content Delivery Server 和任何外部订户数据库中创建订户帐户。	apiContext, username, password, modelId, mobileId, uniqueId, localeCode	subscriberId
signupWithPropertiesAndPreferences	在 Content Delivery Server 和任何外部订户数据库中创建订户帐户，并为该订户设置信息和首选项。	apiContext, username, password, modelId, mobileId, uniqueId, localeCode, propertyList, preferenceList	subscriberId

* 除了列出的对象，所有方法均返回 response_code；如果发生错误则返回 response_message。

11.3.2.11 方法的参数

下表介绍方法的参数。容器对象（例如包含元素的散列表和向量）也在该表中进行介绍。

表 11-10 方法参数

参数	类型	描述
addressLine1	字符串	订户地址的第一行。
addressLine2	字符串	订户地址的第二行。
apiContext	散列表	订户有关信息的容器。该容器包含以下各项： • subscriberId • username • localeCode • mobileId • modelId • uniqueId • isAnonymous • isProvisioned • isRegistered • passwordRequiresReset • isCategoryCustomized • roleId • propertyMap **注 –** 对于匿名订户，仅包含 subscriberId、localeCode 和 modelId。
bundledContentId	整型	包中由 Content Delivery Server 指定给单个项的内部 ID。
bundleId	整型	Content Delivery Server 为当前使用的包指定的内部 ID。
campaign	散列表	活动有关信息的容器。该容器包含以下各项： • campaignId • name • description • campaignSubject • campaignMessage • campaignDiscount • campaignExpiration
campaignDiscount	双精度型	活动中项目的折扣百分比。
campaignExpiration	日期	活动到期日期。如果活动到期，则返回 Null。
campaignId	整型	Content Delivery Server 为当前使用的活动指定的内部 ID。
campaignList	向量，campaign 类型的元素	订户可以使用的活动列表。

表 11-10 方法参数 （续）

参数	类型	描述
campaignMessage	字符串	包含在发送给订户的发布活动通知中的促销消息。
campaignSubject	字符串	添加到发送给订户的发布活动电子邮件通知的主题。
category	散列表	类别有关信息的容器。该容器包含以下各项： • categoryId • name • description • parentCategoryId • isLeafNode • isActive • displayOrder • contentCount • sortingQuery • subCategoryList （仅在由 CategoryHandler 的 getCategoryBranch 方法返回时才包含）
categoryId	整型	Content Delivery Server 为当前使用的类别指定的内部 ID。
categoryIds	向量，整型类型的元素	类别 ID 的列表。类别是由 Content Delivery Server 指定的内部 ID 标识的。
categoryList	向量， category 类型的元素	类别列表。
city	字符串	订户地址的城市。
codedTicket	字符串	标识用于下载请求的购买证明书的字符串。
contactPhone	字符串	订户的电话号码。

表 11-10 方法参数（续）

参数	类型	描述
content	散列表	某项内容有关信息的容器。该容器包含以下各项： • contentId • contentClassId • name • shortDescription • longDescription • submitDate • sizeInKB • version • contentType • isBundle • isBundledItem • mimeType • numberOfDownloads • smallIconUrl • largeIconUrl • userGuideUrl • previewListMap • customFieldCollection • developerName • developerUrl • downloadTimes • networks • downloadUrl • isBookmarked • isPurchaseRequiredBeforeDownload • isUnsubscribeAvailable • isValidOnCurrentModel • pricingDetails • isActive
contentBinary	字符串	内容的二进制格式。
contentClassId	整型	在 Catalog Manager 中标识内容项的唯一标识符。类 ID 是指一个项及其所有关联的版本。
contentCount	整型	类别中内容的项目数。
contentId	整型	Content Delivery Server 为当前使用的内容项指定的内部 ID。
contentIdList	向量，contentId 类型的元素	内容 ID 的列表。

表 11-10 方法参数 （续）

参数	类型	描述
contentKeyword	字符串	用于标识内容的关键字，通常用于使用 SMS 直接传送到设备的内容。
contentLength	整型	内容的大小。
contentList	向量	内容项的列表。在由 ContentHandler 的 getContentDetailsList 方法返回时，向量元素的类型为 content。在由 ContentHandler 的其他方法返回时，向量元素的类型为 contentSummary。
contentSummary	散列表	内容项有关概要信息的容器。该容器包含以下各项： • contentId • name • contentType • shortDescription • submitDate • contentClassId • longDescription • sizeInKB • developerName • contentKeyword • isBundle
contentType	字符串	所使用的内容类型。该类型必须是 contentTypeList 中定义的某一项内容类型。 在由 DownloadHandler 的方法返回时，这是二进制数据或描述符的 MIME 类型。
contentTypeIdList	向量，整型类型的元素	内容类型 ID 的列表。内容类型是由 Content Delivery Server 指定的内部 ID 标识的。
contentTypeList	向量，散列表类型的元素	有关 Content Delivery Server 中定义的每种内容类型的信息。每个元素均包含以下各项： • id • name
country	散列表	国家/地区有关信息的容器。该容器包含以下各项： • countryCode • name
countryCode	字符串	表示订户国家/地区的两字符 ISO 代码，例如，US。
countryList	向量，country 类型的元素	国家/地区的列表。
couponCode	字符串	标识订户用来购买内容的礼券的字符串。

表 11-10 方法参数 （续）

参数	类型	描述
criteria	散列表	内容项有关信息的容器。该容器包含以下各项： • filter • campaignId （可选） • bundleId （仅当内容是包的一部分时） • isSkipTrial • licenseType
customField	散列表	包含为 Content Delivery Server 安装定义的定制字段相关信息的容器。该容器包含以下各项： • customFieldId • customFieldCustomKey • customFieldNumber • customFieldText • customFieldTimestamp • customFieldBoolean • customFieldLabel • customFieldDatatype • customFieldScope
customFieldBoolean	布尔型	为布尔型类型的定制字段存储的值。
customFieldCollection	向量，customField 类型的元素	为安装定义的定制字段列表。
customFieldCustomKey	字符串	用于向 Content Delivery Server 标识定制字段的字符串。
customFieldDatatype	字符串	定制字段中存储的数据的类型。有效值为 boolean、number、 text 和 timestamp。
customFieldId	整型	Content Delivery Server 为标识定制字段而指定的内部 ID。
customFieldLabel	字符串	用于在用户接口中标识定制字段的字符串。
customFieldNumber	双精度型	为 number 类型的定制字段存储的值。
customFieldScope	字符串	指定字段是适用于内容项，还是适用于内容项的版本。
customFieldText	字符串	为 text 类型的定制字段存储的值。
customFieldTimestamp	日期	为 timestamp 类型的定制字段存储的值。
deliveryType	字符串	用于传送内容的传送类型。有效值为： • ems - 增强消息传送系统 • nsm - Nokia 智能消息传送 • one_step_wap - 一步 WAP • two_step_wap - 两步 WAP

表 11-10 方法参数 （续）

参数	类型	描述
description	字符串	对象的描述。根据所调用的方法，该参数可以是类别、活动、设备或语言环境的描述。
descriptorData	字节数组	内容描述符的二进制形式，表示为采用 Base64 编码的数据。在由 DownloadHandler 的 downloadContent 方法返回时，这是内容二进制文件。
developerName	字符串	提交内容的开发者姓名。
developerUrl	字符串	提交内容的开发者 URL。
devicePhone	字符串	订户设备的电话号码。
displayOrder	整型	类别在类别列表中的位置。 1 表示列表顶端。
downloadCount	整型	每一次购买所允许的下载次数。
downloadPeriod	整型	每一次购买所允许使用的天数。
downloadPrice	双精度型	下载内容收取的费用。
downloadTimes	向量，字符串类型的元素	下载内容所需的估计时间。每个元素对应于 networks 对象中的元素，表示通过相应类型网络进行下载所估计的时间。
downloadUrl	字符串	下载内容的 URL。
emailAddress	字符串	订户的电子邮件地址。
eventType	字符串	所使用的事件类型。使用 sms_request_for_content 来表示事件是有关内容项信息的请求。

表 11-10 方法参数（续）

参数	类型	描述
filter	散列表	表示返回信息类型的布尔标志的容器。仅当标志包含在散列表中并设置为 true 时，与每个标志相关联的类型信息才会被返回。 使用 ContentHandler 时，以下标志有效： • filterDetailsResourceBinaries - 预览文件的二进制文件 • filterDetailsDownload - 下载信息 • filterDetailsDownloadCount - 下载计数 • filterDetailsIsBookmarked - 信息书签 • filterDetailsResourceURLs - 预览文件的 URL • filterDetailsPricingAndPurchase - 价格和购买信息 • filterDetailsPricingAndGifting - 礼品的价格信息 • filterDetailsIsValidOnCurrentModel - 有关内容是否在设备上执行方面的信息 • filterDetailsIncludeRetailPrice - 零售价格 **注 –** 如果在过滤器中未指定 filterDetailsResourceBinaries，则预览对象中没有预览文件的二进制文件。如果尝试从预览对象中获取二进制文件，则会返回 Null。如果在过滤器中未指定 filterDetailsResourceURLs，则预览对象中没有预览文件的 URL。如果尝试从预览对象中获取 URL，则会返回空字符串。如果过滤器中不包含这两个标志，则 previewListMap 为 Null 并且无法访问预览文件。 使用 GiftHandler 时，以下标志有效： • filterGiftsContent - 有关内容的信息 • filterGiftsDownload - 下载信息
firstName	字符串	订户的名字。
gender	字符串	订户的性别。有效值以及每个值所表示的含义如以下列表中所述： • M - 订户是男性。 • F - 订户是女性。
giftCost	双精度型	礼品的价格。
giftedDownloads	整型	礼品支付的下载数。
giftedSubscriptions	整型	礼品支付的订阅数。
gifterId	整型	购买礼品的订户的订户 ID。
gifterMobileId	字符串	购买礼品的订户的移动 ID。

表 11-10 方法参数 （续）

参数	类型	描述
giftExpirationDate	日期	礼品到期日期。
giftId	整型	Content Delivery Server 为当前使用的礼品指定的内部 ID。
gifting	散列表	礼品有关信息的容器。该容器包含以下各项： • giftId • giftStatus • giftIsOnDevice • contentId • giftTicket • gifterId • gifterMobileId • giftRecipientId • giftRecipientMobileId • giftRecipientModelId • message • giftedDownloads • giftedSubscriptions • giftCost • giftPurchaseDate • giftExpirationDate • pricingDetails
giftIsOnDevice	布尔型	表示目标设备上是否存在礼品提供内容的标志。 True 表示内容在设备上。 False 表示内容不在设备上。
giftingList	向量，gifting 类型的元素	礼品的列表。
giftPurchaseDate	日期	购买礼品的日期。
giftRecipientId	整型	礼品接收订户的订户 ID。
giftRecipientMobileId	字符串	礼品接收订户的移动 ID。
giftRecipientModelId	整型	礼品接收订户的设备型号 ID。
giftStatus	整型	礼品状态。有效值以及每个值所表示的含义如以下列表中所述： • 8 - 礼品已购买 • 9 - 接收者已开始下载礼品 • 10 - 礼品已成功下载 • 11 - 礼品已过期 • 12 - 礼品已取消 • 13 - 礼品已退款

表 11-10 方法参数（续）

参数	类型	描述
giftTicket	字符串	内部对象，用于验证订户是否可以访问礼品。
hasPurchases	布尔型	表示订户是否已购买内容的标志。 True 表示订户已购买了内容。 False 表示订户未购买内容。
id	整型	Content Delivery Server 为当前使用的语言环境、内容类型指定的内部 ID。
includeContentCount	布尔型	表示是否计算类别中的项数的标志。 True 表示计算该数目。 False 表示不计算该数目。
isActive	布尔型	包含在 category 对象中时，此标志表示是否将类别显示给订户。 True 表示显示类别。 False 表示不显示类别。 包含在 content 对象中时，此标志表示内容是否处于活动状态。 True 表示内容处于活动状态。 False 表示内容处于不活动状态。
isAnonymous	布尔型	表示订户是否为匿名的标志。 True 表示订户为匿名。 False 表示订户为已知。
isAuthenticated	布尔型	此标志表示尝试访问流式传输内容的人员是否为可信的 Content Delivery Server 订户，或者该人是否购买了所请求的项。
isBookmarked	布尔型	表示订户是否已为内容编制书签的标志。 True 表示已为内容编制书签。 False 表示没有为内容编制书签。
isBundle	布尔型	表示该项是否为包的标志。 True 表示该项为包。 False 表示该项不是包。
isBundledItem	布尔型	表示该项是否包含在包中的标志。 True 表示该项是包的一部分。 False 表示该项不是包的一部分。
isCategoryCustomized	布尔型	表示订户是否已定制显示的类别的标志。 True 表示已定制类别。 False 表示未定制类别。
isContentInCampaign	布尔型	表示是否活动中包含内容项的标志。 True 表示活动中包含该项。 False 表示活动中不包含该项。
isDefault	布尔型	表示是否该设备为默认设备的标志。 True 表示该设备是默认设备。 False 表示该设备不是默认设备。
isDownloaded	布尔型	表示是否已下载内容的标志。 True 表示已下载内容项。 False 表示尚未下载内容项。
isDownloadRecurring	布尔型	表示订户是否在超出已购买下载数量之后，自动支付其他下载的标志。 True 表示自动续订。 False 表示订户必须手动购买其他下载。
isFree	布尔型	表示内容是否为免费的标志。 True 表示内容是免费的。 False 表示必须购买内容。
isGiftExpired	布尔型	表示礼品是否已过期的标志。 True 表示礼品已过期，无法再领取。 False 表示礼品未过期。

表 11-10 方法参数 （续）

参数	类型	描述
isGiftingUsed	布尔型	表示是否接收订户已使用为礼品购买的所有用途的标志。 True 表示购买的所有用途已使用。 False 表示不是所有购买的用途都已使用。
isLeafNode	布尔型	表示类别是否具有子类别的标志。 True 表示类别没有子类别。 False 表示类别具有子类别。
isMMSCapable	布尔型	表示是否可以使用 MMS 将内容发送到设备的标志。 True 表示可以使用 MMS。 False 表示不可以使用 MMS。
isOnDevice	布尔型	表示内容项是否在订户设备上的标志。 True 表示内容项在设备上。 False 表示内容项不在设备上。
isOneStepConfirm	布尔型	表示使用一步还是两步下载的标志。 True 表示一步下载。 False 表示两步下载。
isProvisioned	布尔型	表示 Content Delivery Server 数据库中是否存在该订户的条目的标志。 True 表示条目确实存在。 False 表示订户是匿名的。
isPurchaseRequiredBeforeDownload	布尔型	表示是否必须购买内容才能下载的标志。 True 表示必须首先购买内容。 False 表示可以下载内容。
isPushEnabled	布尔型	表示设备是否为可推送的标志。 True 表示设备是可推送的。 False 表示设备不是可推送的。
isRegistered	布尔型	表示订户是否已在外部订户数据库中注册的标志。 True 表示订户已注册。 False 表示订户未注册。
isSkipTrial	布尔型	表示订户是否选择跳过试用的标志。 True 表示订户选择跳过试用，以便可以立即购买内容。 False 表示订户选择不跳过试用。
isSMSCapable	布尔型	表示是否可以使用 SMS 将内容发送到设备的标志。 True 表示可以使用 SMS。 False 表示不能使用 SMS。
isSubscriptionExpired	布尔型	表示是否订阅已过期的标志。 True 表示订阅已过期。 False 表示订阅未过期。
isSubscriptionRecurring	布尔型	表示是否自动续订的标志。 True 表示自动续订。 False 表示订户必须手动续订。
isTicketValid	布尔型	表示订户是否可以使用证明书来获取内容项的标志。 True 表示订户可以使用证明书。 False 表示订户不可以使用证明书。
isTrialAvailable	布尔型	表示内容是否能以试用方式进行使用的标志。 True 表示允许试用。 False 表示禁止试用。
isUpdateAvailable	布尔型	表示内容是否有可用更新的标志。 True 表示有可用更新。 False 表示无可用更新。
isUnsubscribeAvailable	布尔型	表示是否可以取消订阅内容的标志。 True 表示可以取消订阅。 False 表示没有订阅，或者订阅无法取消。

表 11-10 方法参数（续）

参数	类型	描述
isUsageConsumed	布尔型	表示订户是否已使用了所有购买的内容用途的标志。 True 表示所有购买的用途都已使用。 False 表示不是所有购买的用途都已使用。
isValidOnCurrentModel	布尔型	表示是否可以在订户使用的型号上运行内容的标志。 True 表示内容可以在该型号上运行。 False 表示内容不可以在该型号上运行。
keyword	字符串	查找内容时要匹配的字符串。
languageCode	字符串	表示订户语言的两字符 ISO 代码，例如， en。
largeIconUrl	字符串	指向内容大图标的 URL。
lastName	字符串	订户的姓氏。
licenseType	整型	与内容关联的许可证类型。有效值以及每个值所表示的含义如以下列表中所述： • 0 - 已购买内容。 • 1 - 已将内容作为礼品发送。 • 2 - 已将内容作为礼品接收。
listEnd	字符串	它与 listStart 共同指定了要返回的项的范围。按照字母顺序显示的列表以匹配指定字符串的项结尾。字符串区分大小写。指定为 Null 可在完整列表的结尾处结束。
listStart	字符串	它与 listEnd 共同指定了要返回的项的范围。按照字母顺序显示的列表以匹配指定字符串的项开头。字符串区分大小写。指定为 Null 可在完整列表的开头处开始。
locale	散列表	语言环境信息的容器。该容器包含以下各项： • id • countryCode • languageCode • localeCode • description
localeCode	字符串	表示订户语言环境的字符串，例如， en_US。
localeList	向量， locale 类型的元素	语言环境的列表。
longDescription	字符串	来自信息有关内容的详细描述。
manufacturer	字符串	设备生产商的名称。
manufacturerList	向量， manufacturer 类型的元素	生产商的列表，按字母顺序排序。
maxNumberToSend	整型	在调用中传送的最大项数。如果存在的项数多于指定的数量，则不会发送内容。要传送所有项，请使用 -1。

表 11-10 方法参数 （续）

参数	类型	描述
message	字符串	要发送给订户的消息。
messageCategory	整型	要发送的消息类别。类别 1 到 7 发送到订户，类别 9 发送到 Catalog Manager 管理员。有效值以及每个值所表示的含义如以下列表中所述： • 1 - 消息包含指向有关内容项详细信息的 URL。 • 2 - 消息是源于移动的消息，包含指向有关内容项详细信息的 URL。 • 3 - 消息包含内容二进制代码。 • 4 - 消息包含礼品，并且包含指向有关内容项详细信息的 URL。 • 5 - 消息包含通知，并且包含指向有关内容项详细信息的 URL。 • 6 - 消息包含密码提醒。 • 7 - 消息包含有关活动的信息。 • 9 - 将新设备添加到 Content Delivery Server 中。
messageType	整型	要发送消息的类型。有效值以及每个值所表示的含义如以下列表中所述： • 1 - 将消息发送到订户设备。 • 2 - 将消息发送到订户电子邮件。
middleName	字符串	订户的中间名。
mimeType	字符串	内容的 MIME 类型。
mobileId	字符串	订户的电话号码。
model	散列表	设备信息的容器。该容器包含以下各项： • modelId • name • description • modelNumber • manufacturer • userAgentPattern • isDefault
modelId	整型	Content Delivery Server 为当前使用的设备指定的内部 ID。
modelList	向量，model 类型的元素	设备的列表。
modelNumber	字符串	与设备相关的型号。
name	字符串	对象的名称。根据方法，该参数可以是类别、国家/地区、活动、设备或内容的名称。

表 11-10 方法参数 （续）

参数	类型	描述
networks	向量，字符串类型的元素	Content Delivery Server 已知的网络类型的列表。每个元素都对应 downloadTimes 向量中的元素，表示通过相应网络进行下载所估计的时间。
notifyPromotions	布尔型	表示订户是否希望接到促销通知。 True 表示订户希望被通知。 False 表示订户不希望被通知。
numberOfDownloads	整型	内容被所有订户下载的总次数。
numberToReturn	整型	列表中返回的项目数。要返回所有项，请指定 -1。
parentCategoryId	整型	Content Delivery Server 为当前使用类别的父类别指定的内部 ID。
password	字符串	订户不加密的密码。由 Content Delivery Server 执行加密。
passwordRequiresReset	布尔型	表示是否当订户登录时必须重置订户密码的标志。 True 表示必须重置密码。 False 表示无需重置密码。
postalCode	字符串	订户地址的邮政编码。
preferenceList	散列表	订户首选项的容器。此容器包含类型为 notifyPromotions 的项。 要删除某个首选项，将相应的值设置为空字符串 (‘’)。
preview	散列表	预览相关信息的容器。该容器包含以下各项： • previewBinary • previewCaption • previewLastModifiedDate • previewMimeExtension • previewMimeType • previewPositionIndex • previewTarget • previewUrl
previewBinary	字节数组	预览文件的二进制形式，表示为采用 Base64 编码的数据。
previewCaption	字符串	用于标识预览的字符串。此字符串是可选的，并且可以为空。如果需要，代码可能会将空字符串替换为所选的字符串。
previewLastModifiedDate	日期	上次更改预览文件的日期。如果正在缓存预览文件，可以使用该日期来确定是否需要更新缓存。

表 11-10 方法参数 （续）

参数	类型	描述
previewListMap	散列表	WAP 预览和 Web 预览的预览文件的容器。该容器包含以下各项： • wapPreviews • webPreviews 如果在 filter 中未指定 filterDetailsResourceBinaries 和 filterDetailsResourceURLs，则该项为 Null。如果指定了其中一个标志，并且其中的一个目标没有预览文件，则该目标的列表为空。
previewMimeExtension	字符串	原始预览文件的扩展名，例如， .jpg。
previewMimeType	字符串	预览文件的 MIME 类型，例如， image/jpg。
previewPositionIndex	整型	预览集中文件的位置。预览集中的第一个文件的位置索引为 1。
previewTarget	整型	预览文件针对的浏览器类型。有效值以及每个值所表示的含义如以下列表中所述： • 0 - 目标为 WAP 浏览器。 • 1 - 目标为 Web 浏览器。
previewUrl	字符串	指向预览文件的 URL。
pricingDetails	散列表	内容项价格信息的容器。该容器包含以下各项： • campaignDiscount • downloadPrice • downloadCount • downloadPeriod • isDownloadRecurring • subscriptionPrice • subscriptionFrequency • isSubscriptionRecurring • trialCount • usagePrice • usageCount • isFree • isTrialAvailable

表 11-10 方法参数（续）

参数	类型	描述
propertyList	向量，散列表类型的元素	有关每个订户的信息。每个元素均包含以下各项： • salutation • firstName • middleName • lastName • gender • emailAddress • addressLine1 • addressLine2 • city • stateProvince • postalCode • countryCode • contactPhone • devicePhone
propertyMap	散列表	配置系统行为的名称-值对的集合。这些值是 Content Delivery Server 使用的内部值。
purchaseDate	日期	订户购买项目的日期。
purchaseList	向量，散列表类型的元素	有关订户购买的每项内容的信息。每个元素均包含以下各项： • contentId • name • purchaseDate • subscriptionExpiration • pricingDetails • isValidOnCurrentModel • isSubscriptionExpired • isUsageConsumed • isUpdateAvailable • isDownloaded • isOnDevice • codedTicket
recipientApiContext	散列表	礼品接收订户有关信息的容器。该容器包含以下各项： • subscriberId • username • localeCode • mobileId • uniqueId • modelId

表 11-10 方法参数 （续）

参数	类型	描述
requestHeaders	散列表	与 HTTP 请求相关联的 HTTP 头的容器。
requestParams	散列表	提供事件相关信息的键-值对的容器。必须包含以下键： • request_data - 请求中包含的数据，如内容或活动的未解析的 SMS 请求 • request_source - 请求源，如将其发送到的短代码 • request_type - 标识请求类型的字符串，如 portal、mo_push 或系统识别的其他值 也可以包含系统识别的其他键。
response_code	字符串	表示是否成功执行方法的代码。 1 表示成功完成。 -1 表示发生错误。
response_message	字符串	方法返回的消息。
roleId	整型	指定给订户的角色。有效值以及每个值所表示的含义如以下列表中所述： • 0 - 订户仅可以访问状态为 Testing 的内容。 • 1 - 订户具有标准权限。
salutation	字符串	尊称，例如先生。
score	双精度型	该值表示找到的项目与查找查询的相关性有多大。值越大，表示相关性越大。有效值为 0.0 到 1.0。默认值为 0.5。
searchFilter	散列表	用于过滤内容查找结果的条件的容器。该容器包含以下各项： • categoryId • developerName • keyword 容器必须包含至少一种类型的条件。
searchQuery	字符串	Apache Lucene 项目使用的查找查询表达式语言所定义的任何有效查找查询。有关其他信息，请参见 http://lucene.apache.org/java/docs/queryparsersyntax.html。
shortDescription	字符串	来自信息有关内容的简短描述。
sizeInKB	字符串	内容的大小。
smallIconUrl	字符串	指向内容小图标的 URL。
smsParams	散列表	将内容二进制代码推送到设备所需的参数的容器。散列表中的条目是字符串对，用于标识所需的每个参数的名称和值。
sortingQuery	字符串	对查询结果进行排序时所依据的属性。这是 Vending Manager 管理员为某个类别设置的值。如果没有为该类别设置任何值，则使用最近的父类别的查询排序值。如果父类别未指定查询排序值，则使用 null。

表 11-10 方法参数（续）

参数	类型	描述
sortKeyMap	散列表	对查找结果进行排序时依据的字段的容器。散列映射中的条目为字符串和布尔型对，其中字符串为排序字段的名称，布尔型表示是按升序还是按降序排序。值 true 表示升序；false 表示降序。
startIndex	整型	在项目列表中开始进行处理的位置。
stateProvince	字符串	订户地址的省/自治区/直辖市。
status	字符串	确认设备返回的状态字符串。有关 MIDP 应用程序可能返回的值，请参见 http://www.jcp.org/jsr/detail/118.jsp。
statusList	向量，整型类型的元素	内容状态的列表。有效值以及每个值所表示的含义如以下列表中所述： • 1 - 内容处于活动状态。已储存内容，并且订户可以使用。 • 2 - 内容处于不活动状态。已储存内容，但是订户无法使用。 • 3 - 内容不可用。无法再从 Catalog Manager 获取内容。 • 4 - 正在测试内容，只有指定了测试角色的订户才可以使用。
subCategoryList	向量，category 类型的元素	指定节点下面的类别列表。
subject	字符串	要发送给订户的消息主题。
submitDate	日期	提交内容的日期。
subscriberId	整型	Content Delivery Server 为订户帐户指定的内部 ID。
subscriptionExpiration	日期	订户订阅的到期日期。
subscriptionFrequency	字符串	订阅持续的时间段。有效值是 daily、 weekly、monthly 和 yearly。
subscriptionPrice	双精度型	订阅内容需支付的价格。
ticket	字符串	内部对象，用于验证订户是否可访问所请求的内容。
totalSize	整型	找到的项目数。
trialCount	整型	在提示订户购买之前可以免费使用内容的次数。
uniqueId	字符串	订户的唯一 ID。
url	字符串	发送给订户的消息中要包含的 URL。
usageCount	整型	每一次购买所允许使用的次数。
usagePrice	双精度型	每次使用或多次使用支付的价格。
userAgent	字符串	设备的用户代理。该字符串是在 HTTP 头中返回的确切字符串。

表 11-10 方法参数 （续）

参数	类型	描述
userAgentPattern	字符串	设备的用户代理。该字符串是一个正则表达式，是可以匹配各种文本字符串的模式。
userGuideUrl	字符串	指向该内容用户指南的 URL。
username	字符串	订户的用户名。
version	字符串	内容的版本。
wapPreviews	向量，preview 类型的元素	针对 WAP 浏览器的预览文件列表。如果没有针对 WAP 浏览器的预览文件，则返回空列表。
wasDelivered	布尔型	表示是否使用 SMS 将内容发送到设备的标志。 True 表示使用 SMS。 False 表示不使用 SMS。
webPreviews	向量，preview 类型的元素	针对 Web 浏览器的预览文件列表。如果没有针对 Web 浏览器的预览文件，则返回空列表。

11.3.3 使用处理程序的示例

本节提供两个使用订户 API 的 XML-RPC 实现的示例。

11.3.3.1 API 上下文对象创建示例

以下节选代码说明了如何创建 API 上下文对象。该样例使用 Java 编程语言的绑定。

编码样例 11-3 使用 XML-RPC 创建 API 上下文对象

```
...
// Get a reference to the XmlRpcClient
String url = "http://host1:8080/subscriber/xml_rpc.do";
XmlRpcClientLite client = new XmlRpcClientLite(url);

// Set up the input parameters
Vector parameters = new Vector();
Hashtable ht = new Hashtable();
ht.put("username", "user1");
ht.put("password", "cryptic1");
parameters.addElement(ht);

// Send the request to Content Delivery Server
Hashtable response =
(Hashtable) client.execute("AuthenticationHandler.getApiContext", parameters);

// Evaluate the response
String errorCode = (String)response.get("response_code");
```

编码样例 11-3 使用 XML-RPC 创建 API 上下文对象 （续）

```
if (!errorCode.equals("1"))
{
  // Handle Error
  System.out.println((String)response.get("response_message"));
  ...
}
else
{
  // Authentication successful
  Hashtable apiContext = (Hashtable)response.get("apiContext");
  Integer subscriberId = (Integer)apiContext.get("subscriberId");
  String username = (String)apiContext.get("username");
  String localeCode = (String)apiContext.get("localeCode");
  String mobileId = (String)apiContext.get("mobileId");
  Integer modelId = (Integer)apiContext.get("modelId");
  ...

  // Save the ApiContext Hashtable in the Session
  session.setAttribute("API_CONTEXT", apiContext);
  ...
}

  ...
```

11.3.3.2 创建处理程序以及购买内容的示例

以下节选代码显示如何创建处理程序以及使用该处理程序购买内容。该样例使用 Java 编程语言的绑定。

编码样例 11-4 创建处理程序

```
...
// Get a reference to the XmlRpcClient
String url = "http://host1:8080/subscriber/xml_rpc.do";
XmlRpcClientLite client = new XmlRpcClientLite(url);

// Retrieve the ApiContext Hashtable from the HttpSession
Hashtable apiContext = (Hashtable) session.getAttribute("API_CONTEXT");

// Set up the input parameters
Vector parameters = new Vector();
Hashtable ht = new Hashtable();
ht.put("apiContext", apiContext);
ht.put("contentId", new Integer(1001));
ht.put("campaignId", new Integer(1000));
ht.put("isSkipTrial? Boolean.TRUE);
```

编码样例 11-4 创建处理程序 （续）

```
parameters.addElement(ht);

// Send the request to Content Delivery Server
Hashtable response =
(Hashtable) client.execute("ContentHandler.purchaseContent", parameters);

// Evaluate the response
String errorCode = (String)response.get("response_code");
if (!errorCode.equals("1"))
{
  // Handle Error
  System.out.println((String)response.get("response_message"));
  ...
}
else
{
  // Purchase successful
  ...
}
...
```

第 12 章

Vending Manager API

可以使用 Vending Manager API 访问由 Content Delivery Server 维护的 Vending Manager 数据。可以使用该 API 创建和管理包、活动、订户群和订户计划，而无需通过 Vending Manager 管理控制台界面来实现。

可以通过 XML-RPC （Remote Procedure Call, 远程过程调用）实现来访问 Vending Manager API。通过 XML-RPC，应用程序可以在进行传输和数据编码时分别使用 HTTP 和 XML 进行远程过程调用。通过使用 Internet 上提供的绑定，可以将 XML-RPC 与许多不同编程语言一起使用。

注 – 有关 XML-RPC 的教程不在本文档的范围之内。您可以从 Internet 的多个不同 Web 站点中获取有关编写使用 XML-RPC 的应用程序方面的信息。

Vending Manager API 包含以下类：

- `AuthenticationHandler` - 验证用户是否有权访问 Vending Manager 以及处理登录和注销请求。
- `BundleHandler` - 可以用其访问包以及创建、删除和修改包。
- `CampaignHandler` - 可以用其访问活动以及创建、删除和修改活动。
- `CategoryHandler` - 可以用其访问有关类别或类别树的信息。
- `ContentHandler` - 提供类别中的所有内容的列表。
- `DeviceHandler` - 提供 Vending Manager 支持的所有设备的列表。
- `PlanHandler` - 可以用其访问订户计划以及创建、删除和修改订户计划。
- `SubscriberSegmentHandler` - 可以用其访问订户群以及创建、删除和修改订户群。
- `TemplateHandler` - 提供 Vending Manager 中的所有活动模板的列表。
- `UserHandler` - 提供 Vending Manager 已知的所有订户的列表。

有关这些类及其方法和参数的信息，请参见 `$CDS_HOME/javadoc/vendingapi/index.html` 中的 Javadoc 工具的 HTML 输出。

12.1 一般处理流程

可以使用 Vending Manager API 访问 Vending Manager 数据，因此，使用该 API 的任何应用程序必须先通过 `AuthenticationHandler.logUserIn` 方法登录到 Vending Manager。用于登录的用户名必须是有效的 Vending Manager 管理员帐户，而不是客户服务代理。

注 – 所使用的帐户密码不能是默认密码 `admin`。否则，在运行应用程序之前，您必须登录到 Vending Manager 管理控制台并更改该密码。

如果登录成功，则会返回授权密钥。可以在所有后续调用中使用此授权密钥以访问 Vending Manager 数据。在应用程序完成所有任务后，最后一步是使用 `AuthenticationHandler.logUserOut` 方法注销 Vending Manager。

注 – 如果在指定时间 （几分钟）内未使用授权密钥，该密钥将会失效，并且此后的调用将会失败。应用程序必须重新登录以接收新的授权密钥。默认时间为 10 分钟。要更改此时间，请设置 `$CDS_HOME/deployment/`*deployment-name*`/conf/VSAdminConsole.properties` 文件中的 `vendingApi.authkey.timeout.minutes`。

12.2 XML-RPC 方法调用的指导

XML-RPC 调用遵循特定的约定。本节中的样例代码说明了如何进行调用，以及如何处理用户验证结果。该样例代码是使用 Java 编程语言编写的，并使用 Apache XML-RPC 客户机库。如果使用不同的语言或库，需要编写的代码可能会有所不同。

提示 – 有关每种处理程序的方法和参数键的名称，请参见 `$CDS_HOME/javadoc/vendingapi/index.html` 中的 Javadoc 工具的输出。

请按以下指导原则编写处理程序调用代码：

- 将调用的方法参数置于一个散列表中。将散列表置于向量中。

```
...
// Create a vector to hold the request parameters
Vector<Hashtable> request = new Vector<Hashtable>();
```

```
// Create a Hashtable to hold the method specific parameters and add them.
Hashtable<String,Object> methodParams = new Hashtable<String,Object>();
methodParams.put("username", "user1");
methodParams.put("password", "cryptic1");

// Add the method specific parameters to the request.
request.addElement(methodParams);
...
```

- 要调用某种方法，请传递该方法的名称以及包含参数的向量。将返回一个散列表。方法调用必须包含处理程序的名称，例如：AuthenticationHandler.logUserIn。

```
...
// Generate the full method name. ( handlerName.methodName )
String methodName = "AuthenticationHandler.logUserIn";
// Send the request and get the response.
Hashtable response;
response = (Hashtable) client.execute(methodName, request);
...
```

- 通过检查返回的散列表中包含的响应代码，验证方法是否成功执行。如果发生错误，散列表中还包含响应消息。

```
...
// Process the results: check to see if there was an error.
String responseCode = (String)response.get("responseCode");
if(!responseCode.equals(&#xD2;success&#xD3;)) {
   // There was an error. Find out the 'real cause'.
   String realCause = (String)response.get("responseErrorCode");

   System.out.println("Error: Operation failed with: " +realCause);
}
...
```

- 如果方法已成功执行，则提取散列表中返回的值。如果方法未返回任何值，则散列表只包含响应代码。

```
...
// Get the authKey from the request.
String authKey = (String)response.get("authKey");
...
```

下面提供了一个用于处理 Vending Manager 登录请求的完整示例，这是通过 Open Content Delivery Server 项目提供的：
https://opencds.dev.java.net/source/browse/opencds/trunk/modules

/contrib/vendingjavaclient/src/example/com/sun/content/server/vendingjavaclient/xmlrpc/LoginExample.java。该示例仅用于说明目的，本产品并不支持该示例。

12.3 访问 Content Delivery Server

要从 Content Delivery Server 获取数据，应用程序必须能够与 Content Delivery Server 进行通信。请要求网络管理员确保应用程序可以访问 Content Delivery Server，并且任何所需的代理或防火墙均已配置为允许该访问。

此外， Content Delivery Server 还必须认可客户机已授权执行数据请求。`$CDS_HOME/deployment/`*`deployment-name`*`/conf/VSAdminConsole.properties` 文件中的 `vendingApi.xml-rpc.trustedHosts` 属性包含请求将被接受的主机列表。

将 `vendingApi.xml-rpc.trustedHosts` 属性设置为应用程序所在主机的主机名和 IP 地址，而不管是否与 Content Delivery Server 位于同一主机。要接受任何主机的请求，请保留此值为空。要接受多个主机的请求，请使用逗号分隔主机名或 IP 地址，例如:

```
vendingApi.xml-rpc.trustedHosts=127.0.0.1,localhost
```

第 13 章

设备客户机 Web 服务

Web 服务是基于 Web 的应用程序接口，客户机应用程序可通过这些接口访问远程服务中的数据。Content Delivery Server 提供了设备客户机 Web 服务以访问 Content Delivery Server 数据库中存储的信息。可以使用设备客户机 Web 服务验证订户，搜索、预览、购买和下载内容，访问订户的购买历史以及取消内容项订阅。

设备客户机 Web 服务提供了另外一种方法，以访问订户 API 并为订户创建客户机应用程序。该适配器提供了可用服务的概述。有关使用设备客户机 Web 服务的详细信息，请参见 `$CDS_HOME/javadoc/webservices/subscriber/v1` 中的 Javadoc 工具输出。有关配置 Content Delivery Server 以使用设备客户机 Web 服务的信息，请参见《Sun Java™ System Content Delivery Server 5.1 集成和配置指南》中的第 1.11 节。

注 – 有关 Web 服务的教程不在本文档的范围之内。您可以从 Internet 的多个不同 Web 站点中获取有关编写使用 Web 服务的应用程序方面的信息。

可以使用以下 Web 服务：

- 验证
- 目录
- 内容
- 设备
- 历史
- 语言环境

13.1 验证

验证 Web 服务处理验证订户的请求。任何客户机的首要任务是获得访问 Content Delivery Server 数据的授权。对于已知订户和匿名订户，验证 Web 服务将返回一个验证密钥。此密钥用于需要已验证订户的其他设备客户机 Web 服务。匿名用户可以使用设备客户机 Web 服务，但他们并不能使用所有的数据。

还可以使用验证 Web 服务更改订户的首选语言或设备，以及在不需要验证密钥时使其失效。

13.2 目录

目录 Web 服务处理以下请求：浏览或搜索内容以及获取系统中定义的内容类型。可以使用此 Web 服务获取类别中的内容，或者获取与搜索查询匹配的内容。仅返回基于订户计划向订户提供的内容。对于匿名用户，将使用默认订户计划。

13.3 内容

内容 Web 服务处理内容详细信息、预览、资源和二进制文件的请求。此 Web 服务还处理以下请求：购买内容项、在购买之前试用内容项或取消内容项订阅。可以使用此 Web 服务来管理订户与各个内容项和包之间的交互。匿名用户无法购买、试用、下载或取消订阅。

13.4 设备

设备 Web 服务处理以下请求：获取系统中定义的设备以及置备新的设备。可以按系统指定的设备 ID、用户代理或生产商来检索设备。也可以使用此 Web 服务来获取默认设备。

13.5 历史

历史 Web 服务处理订户购买历史的请求。可以使用此 Web 服务来确定订户是否购买了内容项，以及获取所购买的项和包的列表。匿名用户无法使用此 Web 服务。

13.6 语言环境

语言环境 Web 服务处理获取系统中定义的语言环境的请求。可以使用此 Web 服务获取已知语言环境列表以及获取默认语言环境。

索引

E

F

J

K

L

M

N

P

Q

R

S