定制指南

Sun Java™ System Content Delivery Server

版本 2004Q1

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, California 95054
U.S.A.
1-800-555-9SUN 或 1-650-960-1300

文件号码：817-7359
2004 年 8 月

目录

图

表

代码示例

前言

本指南是 Sun Java™ System Content Delivery Server API 的完整参考。其中每一章介绍了用于将 Sun Java System Content Delivery Server 与运营商的现有基础结构集成的一个外部 API。

适用读者

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 Sun Java System Content Delivery Server 与现有基础结构集成的系统管理员，并假设他们已对 Java、联网、数据库和 Web 技术有所了解。

组织

本指南包含下列各章：

- 第 1 章 “简介”概述了 Sun Java System Content Delivery Server 外部 API。
- 第 2 章 “事件服务 API”介绍了提供可记帐事件异步报告的事件服务 API。
- 第 3 章 “记帐 API”介绍了记帐 API，它提供 Content Delivery Server 与记帐系统之间的接口。
- 第 4 章 “内容管理 API”介绍了内容管理 API，它用于管理 Content Delivery Server 与内容管理系统之间的接口。
- 第 5 章 “内容验证 API”介绍了内容验证 API，用于验证和保护提交给 Content Delivery Server 的内容。
- 第 6 章 “用户配置文件 API”介绍了用户配置文件 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 “WAP 网关 API”介绍了 WAP 网关 API，用于从 HTTP 头检索 MSISDN、设备配置文件和其他特性。

- 第 8 章 “消息传送 API”介绍了消息传送 API，它提供了一种机制，使运营商或应用程序供应商能够集成自身的 WAP、 SMS 和 MMS 推送实现。
- 第 9 章 “确认服务 API”介绍了 Content Delivery Server 如何处理从多媒体消息服务中心 (MMSC) 发送的确认消息。
- 第 10 章 “订户 API”介绍了如何访问由 Content Delivery Server 维护的数据。

相关文档

下列文档提供了关于 Sun Java System Content Delivery Server 的信息：

- *Sun Java System Content Delivery Server 容量计划指南*

  本指南适用于负责购买和管理用于 Sun Java System Content Delivery Server 的设备的计划人员和 IT 人员。它提供的指导用于确定所需的硬件和软件。

- *Sun Java System Content Delivery Server 安装指南*

  本指南适用于负责部署和维护 Sun Java System Content Delivery Server 的系统管理员。它提供了有关安装和部署 Content Delivery Server 的信息。

- *Sun Java System Content Delivery Server 集成指南*

  本指南适用于负责配置 Sun Java System Content Delivery Server 以使其在现有基础结构中进行工作的系统管理员。它介绍了用于将 Content Delivery Server 与现有记帐系统、用户数据、 WAP 网关及推送系统集成的适配器，还介绍了创建 Subscriber Portal 的设备特定版本的框架。

- *Sun Java System Content Delivery Server 署名指南*

  本指南适用于可视内容设计人员。它介绍了如何针对自己的企业来定制 Sun Java System Content Delivery Server 的 Subscriber Portal 和 Developer Portal 组件的外观。

- *Sun Java System Content Delivery Server 定制指南*

  本指南适用于负责创建将 Sun Java System Content Delivery Server 与现有基础结构集成时使用的定制适配器的程序员。

- *Sun Java System Content Delivery Server 管理员指南*

  本指南适用于负责管理 Catalog Manager 和 Vending Manager 的系统管理员。它介绍了如何管理内容以及如何定义支持的设备和设备功能。本指南还提供一些说明，用于管理开发者、 Vending Manager 和订户对 Sun Java System Content Delivery Server 的访问。

本指南中使用的惯例

下表总结了本指南中使用的文本特殊处理：

字样	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	输入的内容（相对于计算机屏幕输出信息）。	% **su** Password:
AaBbCc123	书名、新词或术语、要强调的词。	阅读《*用户指南*》中的第 6 章。 这些称为*类*选项。 *必须*是超级用户才能执行此操作。
	命令行变量；使用真实名称或值替换。	要删除文件，请键入 rm *filename*。

访问联机的 Sun 文档

通过 Sun Product Documentation Web 站点可以在 Web 上访问 Content Delivery Server 文档：

http://docs.sun.com

通过 Java Developer Connection[sm] Web 站点可以在 Web 上访问 Java™ 平台技术文档：

http://developer.java.sun.com/developer/infodocs/

Sun 欢迎您提出意见

我们希望能够改进自己的文档，欢迎您提出意见和建议。请将您的意见通过以下地址发送给我们：

http://docs.sun.com/db/form/comments

当您提供意见和建议时，可能需要在表单中提供文档英文版本的标题及文件号码。本文档英文版本的文件号码和标题是：817-4833， Customization Guide。

第 1 章

简介

要部署商业服务，运营商必须将 Sun Java System Content Delivery Server 与现有基础结构集成。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 外部 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
- 用户配置文件 API
- WAP 网关 API
- 消息传送 API
- 确认服务 API
- 订户 API

第 4 页上的第 1.10 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 提供可记帐事件的异步报告，以便外部系统能够提取这些事件并收到事件通知。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，消息的上下文和详细信息都将被提取、插入数据库并传播进行报告和记帐。

可以同时在两个方向传送记帐数据，并且可以在实时模式或批模式下进行数据传送。例如， Content Delivery Server 可以在使用数据被记录时将数据推送到运营商的记帐系统，也可以使用批拉动定期查询服务器。组件之间交换的使用数据是 XML 格式，并包含可以捕获用于记帐集成的价格信息。Content Delivery Server 可以通过此 API 与任何记帐系统进行交互。

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 用于在验证订户帐户资金足够后允许订户下载内容。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程序校验。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对内容进行程序校验时使用此 API。内容管理 API 还可以用于更改内容的特性，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 可创建为提交的内容执行定制自动验证步骤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 用户配置文件 API

用户配置文件 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。运营商不需要创建单独的新数据库来配合 Java System Content Delivery Server 使用。如果运营商需要组合或合并多个数据库项，则用户配置文件 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。

通过用户配置文件 API， Content Delivery Server 为所有访问数据库的组件提供了一个公用服务层。将数据访问抽象化使得 Content Delivery Server 可以不受具体数据库的束缚。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，认证将通过网络执行，且无需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。

Content Delivery Server 使用标准 HTTP Web 服务器处理表示逻辑。它既为订户 Web 站点和管理员控制台提供 HTML 内容，也可以为通过 WAP 1.*x* 兼容的浏览器提供设备端访问的 WML。 Content Delivery Server 支持 SSL/TLS 连接以确保通信的安全。

1.7 消息传送 API

消息传送 API 为运营商或应用程序供应商集成自身的 WAP、SMS 推送实现（通过提供适配器进行集成）提供了一种机制。 Content Delivery Server 还提供了可以在大多数情况下使用的缺省 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 (MMSC) 发送的确认信息。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 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，以便为订户提供对由 Content Delivery Server 管理的内容的访问。订户 API 可以通过基于 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 从远程应用程序或以非 Java 的语言编写的应用程序访问。

1.10 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 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 (SMTP)、短消息服务中心 (SMSC)、推式代理网关 (PPG) 和多媒体消息服务中心 (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
宣传活动	无	无
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	轻型目录访问协议 (LDAP)	用户配置文件 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 (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
门户集成	Web 门户	订户 API

第 2 章

事件服务 API

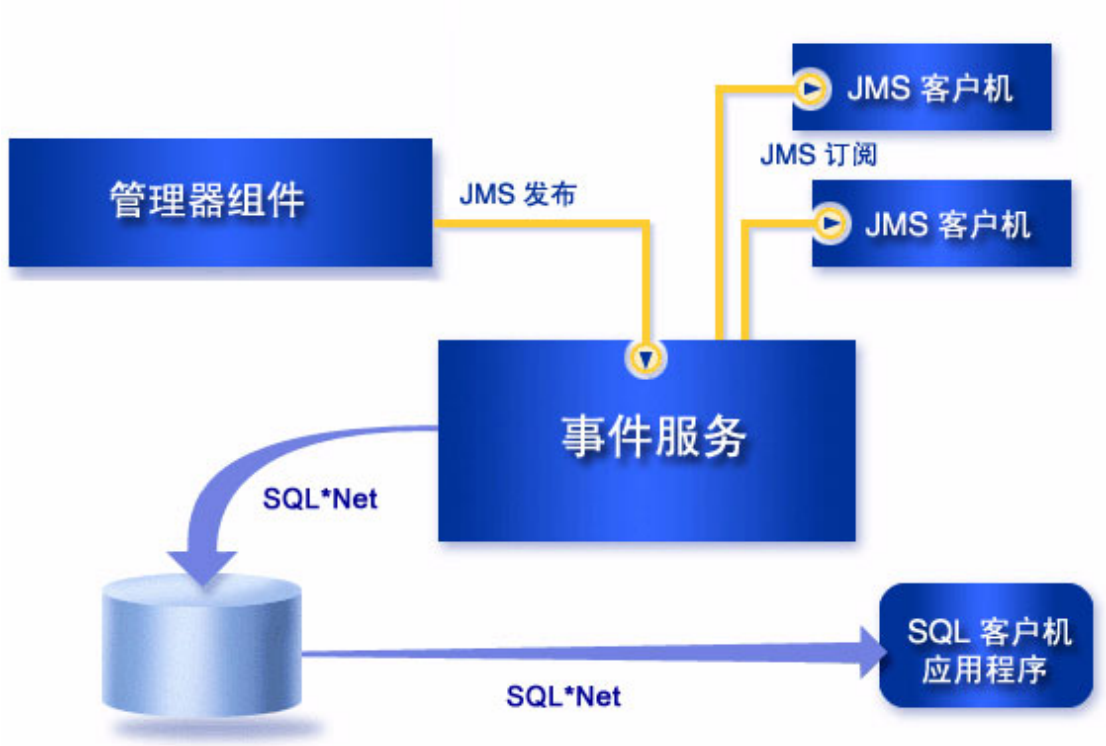
本章介绍了 Sun Java System Content Delivery Server 事件服务 API。事件服务 API 的外部接口是

- 用于通过 SQL*Net 客户机应用程序直接查询事件数据的数据库模式。
- 可以由 JMS 客户机应用程序订阅的 Java 消息服务 (JMS) 主题。

事件服务会将事件队列传出的消息传播给已订阅了事件服务所发布的主题的任何感兴趣的事件监听器。事件服务还将事件数据存储在 Content Delivery Server 数据库中。可以使用 SQL*Net 客户机或 JMS 客户机应用程序方法。两种方法都可以近乎实时地访问相同的信息。

图 1 简单说明了通用系统和组件与事件服务之间进行交互的情况，列出了通过各种系统交互传递信息的方式：

图 1 事件服务概述



Content Delivery Server 管理器将适当的消息发布到 Java 消息服务 (JMS) 队列。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，所以一旦发出消息，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。事件服务用于：

- 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资料。该信息将插入事件数据库中。
- 在成功处理消息后，向 JMS 队列返回确认。
- 如果发生错误，则将消息放入错误队列，从中可以处理错误并可能将其重新提交给事件队列。消息仍确认为由事件服务成功处理。

事件服务 API 的当前实现使用了 JMS 点对点 (PTP) 消息传送域，在此域中各种发布程序将消息发布到单个队列，并由事件服务 JMS 客户机应用程序的某个实例对这些消息进行处理。

事件被事件服务成功处理后，将被放置到用于存放消息内容的数据库架构区域。有关存储这些消息的数据模型信息，请参见第 7 页上的 “事件数据库”。

2.1 SQL*Net 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 语言、 C++ 或 Visual Basic 编写这些应用程序。

2.1.1 事件数据库

本节所介绍的表格包含事件经事件服务处理后产生的数据。作为集成者，您具有访问这些表格数据的权限。使用用户 *prefix*_es_app 可以连接到 Oracle 数据库，其中 *prefix* 是为用于创建数据库的数据库配置文件中 Vending 元素下的 Prefix 元素指定的值。使用为数据库配置文件中的 Vending 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 安装指南*》。

2.1.1.1 CDS_EVENT 表

此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 CDS_EVENT 表

列名	数据类型	说明
CDS_EVENT_ID	NUMBER(18)	用作记录 ID 的系统生成的唯一数字。此字段是表的主关键字。
CDS_EVENT_DATE	DATE	生成事件消息的时间标记。
CDS_EVENT_TYPE_ID	NUMBER(18)	CDS_EVENT_TYPE 表的外关键字。
EVENT_SOURCE_ID	NUMBER(18)	EVENT_SOURCE_TYPE 表的外关键字。
SVR_INSTANCE_ID	NUMBER(18)	系统数据。
SVR_SESSION_ID	VARCHAR2(128)	系统数据。
SUB_SYSTEM_ID	VARCHAR2(80)	与会话及事件关联的 MSISDN。如果 MSISDN 未知，则该列为空。
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，则此列可能为空。
VENDOR_ID	NUMBER(18)	供应商表的外关键字。
CONTENT_ID	NUMBER(18)	内容表的外关键字。
RAW_EVENT_MESSAGE	CLOB	原始事件消息 XML。

表 2 CDS_EVENT 表 *(续)*

列名	数据类型	说明
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

此表包含事件类型定义。在第 10 页上的第 2.3 节 “事件和事件数据”中列出了这些定义。它是静态的表。

表 3 CDS_EVENT_TYPE 表

列名	数据类型	说明和建议
CDS_EVENT_TYPE_ID	NUMBER(18)	用作记录 ID 的系统生成的唯一数字。此字段是表的主关键字。
CDS_EVENT_GROUP_ID	NUMBER(18)	EVENT_GROUP 表的外关键字。用于对事件类型进行分组。
CDS_EVENT_TYPE_NAME	VARCHAR2(80)	可读名称。
DESCRIPTION	VARCHAR2(80)	特定事件类型的说明。
LONG_DESCRIPTION	VARCHAR2(255)	事件类型的详细说明 （如果需要）。
IS_ACTIVE	NUMBER(1)	表明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 CDS_EVENT_GROUP 表

此表保留了系统中所有事件组的定义，为静态形式。所有事件都属于名为 cds_group 的事件组。

表 4 CDS_EVENT_GROUP 表

列名	数据类型	说明和建议
CDS_EVENT_GROUP_ID	NUMBER(18)	用作记录 ID 的系统生成的唯一数字。此字段是表的主关键字。
CDS_EVENT_GROUP_NAME	VARCHAR2(255)	可读名称。
DESCRIPTION	VARCHAR2(1024)	特定事件组的说明。

表 4 CDS_EVENT_GROUP 表 *(续)*

列名	数据类型	说明和建议
LONG_DESCRIPTION	VARCHAR2(2048)	事件组的详细说明 （如果需要）。
PARENT_GROUP_ID	NUMBER(18)	允许使用组分层结构。
IS_ACTIVE	NUMBER(1)	表明是否为活动事件组的标志。
CREATE_DATE	DATE	系统数据。
MOD_DATE	DATE	系统数据。
LOCK_VERSION	NUMBER(1)	系统数据。

2.1.1.4 EVENT_SOURCE_TYPE_ID 表

此表包含事件源类型定义。它是静态的表。

表 5 EVENT_SOURCE_TYPE_ID 表

列名	数据类型	说明和建议
EVENT_SOURCE_TYPE_ID	NUMBER(18)	用作记录 ID 的系统生成的唯一数字。此字段是表的主关键字。
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为数据生成各种类型的报告。常用的一些报告工具包括 Crystal Reports、 Microsoft Access、 Oracle Reports 或 ReportSmith。

2.2 JMS 客户机应用程序

除了事件数据的基于 SQL 的接口，还可以实现通过 JMS 主题与事件服务直接接口的 JMS 客户机应用程序。虽然此方法需要进行较为复杂的开发，但是可以提高处理 Content Delivery Server 生成的事件时的灵活性。

要成功利用事件服务 （使用此 API）集成，应该熟悉 JMS 客户机应用程序的编写，并且理解 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。 SQL*Net 和 JMS 客户机应用程序都使用此信息过滤和处理事件。

下表介绍了由事件服务生成的事件。

表 6 事件

事件	说明
content_changed	Catalog Manager 管理员已对该内容进行了更改。
content_purchased	订户已购买了内容项。
content_refunded	已对内容项发放了退款。
download_deleted	已从设备中删除了下载的内容。
download_error	设备已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	接收人已下载了礼品。
gift_download_deleted	接收人下载的礼品已被删除。
gift_download_error	下载礼品时出错。
gift_download_initiated	礼品接收人已开始下载礼品。
gift_expired	礼品已过期。
gift_purchased	已作为礼品购买了内容项。
gift_subscription_purchased	已作为礼品购买了内容项订阅。

表 6 事件 *(续)*

事件	说明
gift_usage_purchased	已作为礼品购买了对内容项的多次使用权。
mms_push_sent	已发送 MMS 消息。
pricing_changed	一个或多个内容项的价格已使用 “类别价格编辑” 功能进行了更改。
sms_push_sent	已发送 SMS 消息。
sms_received	已收到 SMS 消息。
smtp_push_sent	已发送 SMTP 消息。
status_changed_to_deleted	内容项的状态已更改为 “已删除”。
status_changed_to_denied	内容项的状态已更改为 “已拒绝”。
status_changed_to_new	内容项的状态已更改为 “新建”。
status_changed_to_pending	内容项的状态已更改为 “待定”。
status_changed_to_published	内容项的状态已更改为 “已发布”。
status_changed_to_testing	内容项的状态已更改为 “正在测试”。
submission_failed	提交的内容遭到 Content Delivery Server 拒绝。
submission_successful	提交的内容已被 Content Delivery Server 接受。
subscriber_registered	订户已成功注册。
subscription_cancelled	对内容项的订阅已被取消。
subscription_purchased	对内容项的订阅已被购买。
usage_purchased	已购买了内容项的多次使用权。
validation_passed	提交的内容已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	已发送 WAP 消息。

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 7 事件数据

参数	数据类型	说明
billing-ticket	字符串	该事务的记帐证明书。
campaign_coupon	字符串	活动的礼券代码。
campaign_id	字符串	标识活动的字符串。
catalog-res-id	字符串	标识内容版的字符串。
content_class_id	字符串	标识内容项的字符串。
content_description	字符串	内容的说明。

表 7 事件数据 *(续)*

参数	数据类型	说明
content-id	字符串	用于标识所购买内容的字符串。此值与 catalog-res-id 相同。
content_name	字符串	内容名称。
current-status	字符串	事务的当前状态。
date	日期	事务的发生日期。
developer-content-id	字符串	开发者用于标识内容的字符串。
developer-id	字符串	用于标识内容开发者的字符串。
developer_name	字符串	提交内容的开发者的姓名。
download-confirm	布尔型	用于指明成功下载后是否需要确认的标志。
download-count	整型	根据支付价格确定的内容下载次数。
download-current-count	整型	订户已下载此内容的次数 （包括此次）。
download-expiration	布尔型	用于表明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额外费用的时段。
download-price	浮点	内容的购买价格。
download-purchase	布尔型	用于表示购买请求的标志。
download-recurring	布尔型	用于表明是否应向订户收取每次下载费用的标志。
event-log	字符串	事件日志名称。
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	用于标识已发生事件的类型的字符串。
external_content_id	字符串	用于标识记帐系统内容的字符串。
external_group_id	字符串	用于标识内容所属的组的字符串。
gift_message	字符串	礼品中包含的消息。
gifted_current_downloads	整型	接收订户已下载此礼品的次数 （包括此次）。
gifted_current_subscriptions	整型	接收订户使用的订阅时段数 （包括此时段）。
gift_download_date	日期	接收订户第一次下载礼品的日期。
gift_expiration_date	日期	接收订户认领礼品的截止日期。
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	用于表明内容是否已存在于设备中的标志。

表 7 事件数据 *(续)*

参数	数据类型	说明
is-prepay	布尔型	用于表明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。
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	内容的预定接收订户的语言环境。
recipient_login_id	字符串	内容的预定接收订户的登录 ID。
recipient_mobile_id	字符串	内容的预定接收订户的移动 ID。
recipient_unique_device_id	字符串	预定接收订户的唯一设备 ID。
server-id	字符串	用于标识 Vending Manager 的字符串。
session-id	字符串	用于标识订户会话的字符串。
subscription-expiration	日期	订阅时段截止日期。
subscription-frequency	字符串	收取订阅费用的时间间隔。
subscription-recurring	布尔型	用于表明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	订阅时段的价格。
timestamp	时间戳	事务的发生时间。
unique-device-id	字符串	用于唯一标识所用设备的字符串。
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	根据为 usage-count 指定的使用次数而确定的收费。
user-id	字符串	用于标识启动事务的用户的字符串。
username	字符串	订户的登录名。
vending-res-id	字符串	Vending Manager 用于标识内容的字符串。

2.4 CDSAbstractBillingSubscriber 类

`CDSAbstractBillingSubscriber` 类是抽象类，为所有订户类的基类。它用于处理所有与 JMS 相关的信息。这样外部开发者便可以集中精力进行自身的商业逻辑开发。

有关本节中未介绍的类或方法的信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 中的事件服务 API 的 Javadoc™ 工具的 HTML 输出。

2.4.1 构造函数

在您实现的构造函数中，调用父级构造函数并提供表 8 中所述的参数。执行该实现时，可以对这些参数进行硬编码或传递这些参数。

```
CDSAbstractBillingSubscriber(java.lang.String clientID,
    java.lang.String subName, java.lang.String selector,
    boolean durable, boolean transactional,
    java.lang.String jndiCtxFactory, java.lang.String connFactory,
    java.lang.String providerURL, java.lang.String topicName)
```

表 8 构造函数参数

参数	说明
clientID	用于标识连接的名称。
subName	用于标识 JMS 主题订阅的名称。
selector	用于确定所监视的事件的过滤器。将此参数设置为 **`null`** 可以监视所有事件。要监视特定事件，请参见 `javax.jms.Message` 的 Javadoc 工具的输出，以获取有关设置此参数的信息。有关事件信息，请参见第 7 页的表 2。
durable	表示订户是否为长期订户的标志。长期订户接收订户处于非活动状态时发送的消息。将此参数设置为 **`true`** 可以创建长期订户。否则，将其设置为 **`false`**。
transactional	指明事务性会话是否应用于创建订户的标志。将此参数设置为 **`true`** 可以创建事务性会话。否则将其设置为 **`false`**。
jndiCtxFactory	用于生成 WebLogic 初始上下文的类。如果使用的是 Sun Java System Application Server，则将此参数设置为 **`null`**。如果使用的是 WebLogic Server，则将此参数设置为 **`weblogic.jndi.WLInitialContextFactory`**。

表 8 构造函数参数 *(续)*

参数	说明
connFactory	Java Naming and Directory Interface™ ("JNDI") API 使用的连接工厂的名称。将此参数设置为 **cds.messaging.billing.TopicConnectionFactory**。
providerURL	JNDI API 的服务提供者的 URL。 如果使用的是 Sun Java System Application Server，则将此参数设置为 **null**。 如果使用的是 WebLogic Server，则为 `jndi.host` 和 `jndi.port` 属性，将此参数设置为部署配置文件中的值。例如， **t3://***jndihost.domain.com***:***80*。
topicName	JNDI API 使用的 `CDSBillingTopic` 主题的名称。将此变量设置为 **cds.messaging.billingTopic**。

2.4.2 handleMsg()

此方法用于定义如何处理消息。它由 `onMessage` 进行调用。

```
public abstract void handleMsg()
```

2.4.3 handleError()

此方法用于定义如何在消息处理过程中处理错误。它由 catch 块中的 `onMessage` 调用。

```
public abstract void handleError()
```

2.5 使用事件服务 API

本节介绍如何将事件服务 API 用于 SQL*Net 和 Java 中。

2.5.1 开发 SQL*Net 客户机应用程序

应用程序必须获得指向数据库的 SQL*Net 连接，以便应用程序直接查询 Oracle 数据库。具体过程取决于数据库服务器和客户机应用程序的本地环境。

使用用户 *prefix*_es_app 可以连接到 Oracle 数据库，其中 *prefix* 是为用于创建数据库的数据库配置文件中 Vending 元素下的 Prefix 元素指定的值。使用为数据库配置文件中的 Vending 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 安装指南*》。

2.5.2 开发 JMS 客户机应用程序

要编译 JMS 客户机，请在类路径中包含下表中所述的文件：

表 9 编译客户机所需的文件

JAR 文件	位置 [1]
JMS Java 归档 (JAR) 文件	/*cds-home*/deployment/*deployment-name*/lib/jms.jar
Content Delivery Server JAR 文件	/*cds-home*/deployment/*deployment-name*/lib/cdslib/cdsapi.jar
Foundation JAR 文件	/*cds-home*/deployment/*deployment-name*/lib/cdslib/foundation.jar

[1] *cds-home* 是 Content Delivery Server 的安装目录。 *deployment-name* 是对部署赋予的名称。

要执行 JMS 客户机：

- 在类路径中包含下表中所述的文件。

表 10 执行客户机所需的文件

JAR 文件	位置 [1]
JMS JAR 文件	*cds-home*/deployment/*deployment-name*/lib/jms.jar
Content Delivery Server JAR 文件	*cds-home*/deployment/*deployment-name*/lib/cdslib/cdsapi.jar
Foundation JAR 文件	*cds-home*/deployment/*deployment-name*/lib/cdslib/foundation.jar
事件服务客户机 JAR 文件	*cds-home*/deployment/*deployment-name*/lib/cdslib/eventserviceclient.jar
文件系统上下文 JAR 文件（仅当使用 Sun Java System Application Server 时需要）	*cds-home*/deployment/*deployment-name*/lib/cdslib/fscontext.jar
Message Queue JAR 文件（仅当使用 Sun Java System Application Server 时需要）	*cds-home*/deployment/*deployment-name*/lib/cdslib/imq.jar
WebLogic JAR 文件（仅当使用 WebLogic Server 时需要）	*wl-home*/lib/weblogic.jar

[1] *cds-home* 是 Content Delivery Server 的安装目录。 *deployment-name* 是对部署赋予的名称。

- 指定以下选项：
  - `-Dcds.home=`*cds-home*
  - `-Dcds.config.file=CDS.properties`
  - `-Dcds.config.dir=`*cds-home*`/deployment/`*deployment-name*`/conf`

以下代码示例展示了一个样例脚本，可用于编译和执行第 19 页上的第 2.6 节“示例”中显示的 `CDSAbstractBillingSubscriber` 类的样例实现。

代码示例 1 编译和执行的样例脚本

```
#!/bin/bash

# Set the following properties for your environment
CDS_HOME="/usr/local/cdshome"
CDS_DEPL_NAME="cds"
JAVA_HOME="/usr/j2se"

# Set the following parameters for CDSBillingSubscriber
clientID="CDSBillingSubscriberUniqueClientID"
subName="CDSBillingSubscriberUniqueSubscriptionName"
selector="null"
durable="false"
transactional="false"

# CDSBillingSubscriber Parameter Constants
connFactory="cds.messaging.billing.TopicConnectionFactory"
topicName="cds.messaging.billingTopic"

# Sun Java System Application Server Parameters
jndiCtxFactory="null"
providerURL="null"

# BEA WebLogic Parameters
# Set the providerURL and WL_HOME
# For providerURL use the
# WebLogic Application Server host and port
#jndiCtxFactory="weblogic.jndi.WLInitialContextFactory"
#providerURL="t3://jndihost.domain.com:80"
#WL_HOME="/usr/local/bea/weblogic700/server"

# You may select to filter the messages by event.
# See the javadoc for javax.jms.Message for more
# information on selectors.See the CDS Customization
# Guide for a list of possible events.
#selector="JMSCorrelationID=download_initiated"

# Directory Constants
DEPL_DIR="${CDS_HOME}/deployment/${CDS_DEPL_NAME}"
CONF_DIR="${DEPL_DIR}/conf"
LIB_DIR="${DEPL_DIR}/lib"
CDSLIB_DIR="${LIB_DIR}/cdslib"

# Sun Java System Application Server CLASSPATH Constant
COMPILE_LCP=".:${LIB_DIR}/jms.jar"
COMPILE_LCP="${COMPILE_LCP}:${CDSLIB_DIR}/cdsapi.jar"
COMPILE_LCP="${COMPILE_LCP}:${CDSLIB_DIR}/foundation.jar"
EXEC_LCP=".:${LIB_DIR}/jms.jar"
EXEC_LCP="${EXEC_LCP}:${CDSLIB_DIR}/cdsapi.jar"
EXEC_LCP="${EXEC_LCP}:${CDSLIB_DIR}/foundation.jar"
EXEC_LCP="${EXEC_LCP}:${CDSLIB_DIR}/eventserviceclient.jar"
```

```
EXEC_LCP="${EXEC_LCP}:${LIB_DIR}/imq.jar"
EXEC_LCP="${EXEC_LCP}:${LIB_DIR}/fscontext.jar"

# BEA WebLogic CLASSPATH Constant
#COMPILE_LCP=".:${LIB_DIR}/jms.jar"
#COMPILE_LCP="${COMPILE_LCP}:${CDSLIB_DIR}/cdsapi.jar"
#COMPILE_LCP="${COMPILE_LCP}:${CDSLIB_DIR}/foundation.jar"
#EXEC_LCP=".:${LIB_DIR}/jms.jar"
#EXEC_LCP="${EXEC_LCP}:${CDSLIB_DIR}/cdsapi.jar"
#EXEC_LCP="${EXEC_LCP}:${CDSLIB_DIR}/foundation.jar"
#EXEC_LCP="${EXEC_LCP}:${CDSLIB_DIR}/eventserviceclient.jar"
#EXEC_LCP="${EXEC_LCP}:${WL_HOME}/lib/weblogic.jar"

# Java Command Line Option Constants
CDS_OPTS="-Dcds.home=${CDS_HOME}"
CDS_OPTS="${CDS_OPTS} -Dcds.config.file=CDS.properties"
CDS_OPTS="${CDS_OPTS} -Dcds.config.dir=${CONF_DIR}"

# Compile
JAVAC_OPTS="-classpath ${COMPILE_LCP}"
JAVAC_OPTS="${JAVAC_OPTS} CDSBillingSubscriber.java"
$JAVA_HOME/bin/javac ${JAVAC_OPTS}

# Execute
JAVA_OPTS="-cp ${EXEC_LCP} ${CDS_OPTS}"
JAVA_OPTS="${JAVA_OPTS} CDSBillingSubscriber"
JAVA_OPTS="${JAVA_OPTS} ${clientID}"
JAVA_OPTS="${JAVA_OPTS} ${subName}"
JAVA_OPTS="${JAVA_OPTS} ${selector}"
JAVA_OPTS="${JAVA_OPTS} ${durable}"
JAVA_OPTS="${JAVA_OPTS} ${transactional}"
JAVA_OPTS="${JAVA_OPTS} ${jndiCtxFactory}"
JAVA_OPTS="${JAVA_OPTS} ${connFactory}"
JAVA_OPTS="${JAVA_OPTS} ${providerURL}"
JAVA_OPTS="${JAVA_OPTS} ${topicName}"
$JAVA_HOME/bin/java ${JAVA_OPTS}
```

2.6 示例

以下代码示例是 CDSBillingSubscriber 类的类定义。此类是如何扩展 AbstractBillingSubscriber 类以实现自己的记帐适配器的样例。

代码示例 2 CDSAbstractBillingSubscriber 样例实现

```
import com.sun.content.server.eventservice.subscriber.abs.*;
import com.sun.content.server.eventservice.subscriber.util.*;
import javax.jms.TextMessage;

public class CDSBillingSubscriber
extends CDSAbstractBillingSubscriber
{
  public CDSBillingSubscriber(
    String clientID,
    String subName,
    String selector,
    boolean durable,
    boolean transactional,
    String jndiCtxFactory,
    String connFactory,
    String providerURL,
    String topicName)
  throws Exception
  {
    super(clientID, subName, selector, durable, transactional,
      jndiCtxFactory, connFactory, providerURL, topicName);
  }

  public void handleMsg()
  throws CDSEventServiceSubscriberException
  {
    try
    {
      // simply prints the message
      TextMessage txtMsg = (TextMessage) getMessage();
      System.out.println(txtMsg.getText());
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }

  public void handleError()
  throws CDSEventServiceSubscriberException {}

  private static void displayUsage()
  {
    System.out.println();
    System.out.print("Usage :java CDSBillingSubscriber ");
    System.out.print("clientID subName selector ");
    System.out.print("durable transactional ");
    System.out.print("jndiCtxFactory connFactory ");
    System.out.print("providerURL topicName");
    System.out.println();
```

```
  System.out.println();
  System.out.print("For selector, you can specify null or ");
  System.out.print("JMSCorrelationID=<event>");
  System.out.println();
  System.out.println();
  System.out.println("\t Terminating Program ...");
  System.exit(-1);
}

public static void main(String[] args)
{
  String clientID = null;
  String subName = null;
  String selector = null;
  boolean durable = false;
  boolean transactional = false;
  String jndiCtxFactory = null;
  String connFactory = null;
  String providerURL = null;
  String topicName = null;
  if (args.length == 9)
  {
    clientID = args[0];
    subName = args[1];
    selector = args[2];
    durable = Boolean.valueOf(args[3]).booleanValue();
    transactional = Boolean.valueOf(args[4]).booleanValue();
    jndiCtxFactory = args[5];
    connFactory = args[6];
    providerURL = args[7];
    topicName = args[8];
  }
  else
  {
    displayUsage();
  }

  if (selector == null)
    selector = null;
  else if (selector.trim().length() == 0)
    selector = null;
  else if (selector.trim().equalsIgnoreCase("null"))
    selector = null;


  CDSBillingSubscriber s = null;
  try
  {
    System.out.println();
    System.out.println("Creating subscriber...");
    s = new CDSBillingSubscriber(clientID, subName,
        selector, durable,transactional, jndiCtxFactory,
```

```
            connFactory, providerURL, topicName);
        System.out.println("Subscriber created.");
        System.out.println("clientID="+s.getClientID());
        System.out.println("subName="+s.getSubName());
        System.out.println("selector="+s.getSelector());
        System.out.println("durable="+s.isDurable());
        System.out.println("transactional="+s.isTransactional());
        System.out.println("jndiCtxFactory="+s.getJndiCtxFactory());
        System.out.println("connFactory="+s.getConnFactory());
        System.out.println("providerURL="+s.getProviderURL());
        System.out.println("topicName="+s.getTopicName());

        System.out.println();
        System.out.println("Running for ten minutes...");
        Thread.sleep(600000);
        System.out.println("Ten minutes have elapsed.");
      }
      catch (Exception e)
      {
        e.printStackTrace();
      }

      try
      {
        System.out.println();
        System.out.println("Closing connections...");
        if (s != null)
          s.getTopicHelper().closeConnections();
        System.out.println("Connections closed");
      }
      catch (Exception e)
      {
        e.printStackTrace();
      }
    }
  }
```

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 包括以下类：

- `BillingManager` 接口：用于实现将 Content Delivery Server 与记帐系统进行集成的接口。`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` 类：此类包含记帐事务信息，例如内容的价格和定价模型、订户标识、内容标识、开发者标识和记帐状态。此类还包括处理信息，例如订户下载项目之前是否需要授权，以及内容被成功下载后记帐系统是否应收到通知。`BillingInfo` 对象提供记帐系统与 Content Delivery Server 之间的通信。
- `BillingException` 类：记帐 API 抛出的异常。
- `BillingConstants` 类：定义记帐 API 使用的常数的类。

有关这些类的其他信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 处的 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。记帐事务的详细资料保留在 `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。

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

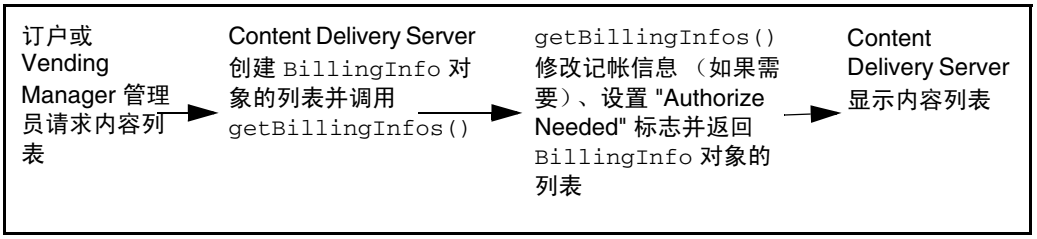
- 内容列表
- 事务启动
- 订户购买
- 下载确认
- 订阅验证
- 错误处理

有关本节涉及的 BillingManager 方法的详细资料，请参见第 28 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 2 显示了此进程。

图 2 内容列表的进程流

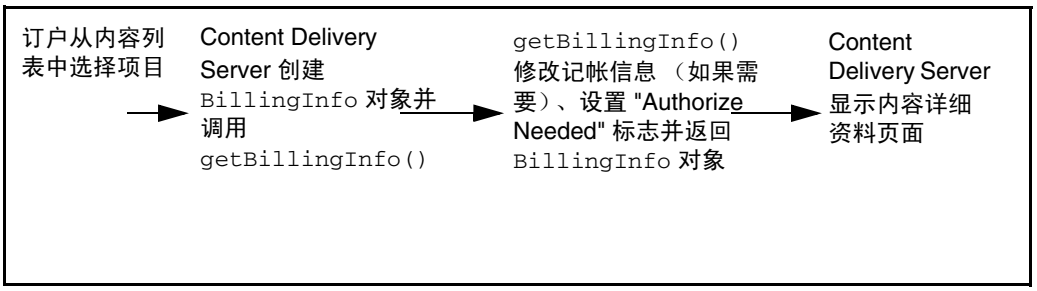


1. 当订户或 Vending Manager 管理员请求内容列表时， Content Delivery Server 将为列表中每个对象创建初始 BillingInfo 对象，并调用记帐适配器的 getBillingInfos() 方法。
2. 如果需要， getBillingInfos() 的实现可以修改购买每个事务的价格或其他详细资料。您也可以指定购买时是否需要授权，以及内容下载成功后是否需要确认消息。这些详细资料在返回到 Content Delivery Server 的 BillingInfo 对象中进行设置。有关其他信息，请参见第 30 页上的第 3.2.5 节 “getBillingInfos()”。
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

当订户单击内容项目旁边的 “**查看详细资料**”时，记帐事务将启动。图 3 显示了此进程。

图 3 事务启动进程流



1. 当订户请求项目的详细资料时，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。

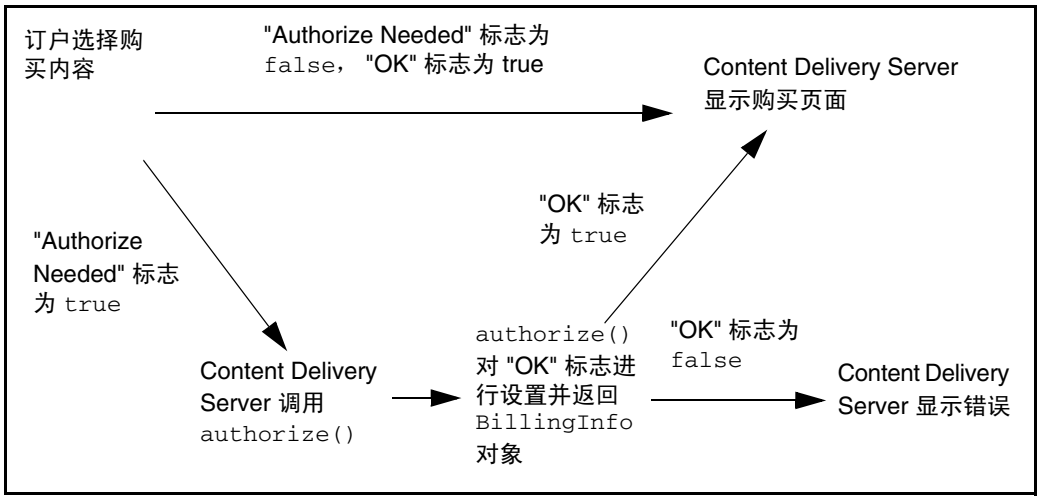
3. 如果需要， `getBillingInfo()` 的实现可以修改购买价格或事务的其他详细资料。您也可以指定购买时是否需要授权，以及内容下载成功后是否需要确认消息。这些详细资料在返回到 Content Delivery Server 的 `BillingInfo` 对象中进行设置。有关其他信息，请参见第 29 页上的第 3.2.4 节 “getBillingInfo()”。

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

当用户选择项目并单击 “**购买**” 时，购买进程将启动。购买进程将使用启动事务时创建的 `BillingInfo` 对象。图 4 显示了此进程。

图 4 订户购买进程流



1. 如果订户选择购买内容，Content Delivery Server 将检查由 `getBillingInfo()` 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。此方法应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 28 页上的第 3.2.1 节 “authorize()”。

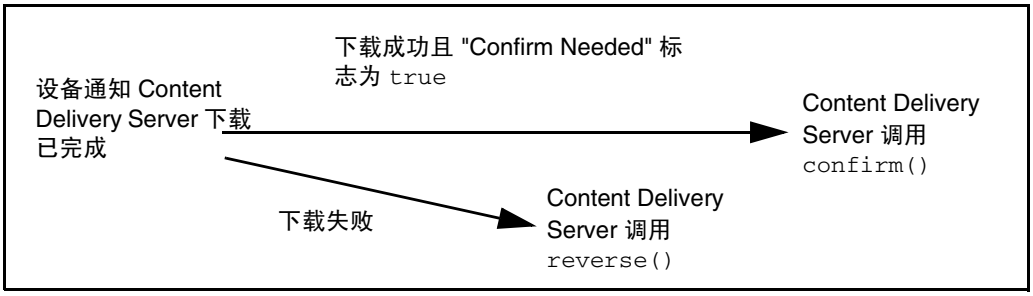
3. 如果订户被授权购买内容或无需授权，则显示购买页面。如果订户未被授权购买内容，则发出错误消息。

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，确认进程将启动。第 26 页的图 5 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 5 下载确认进程流

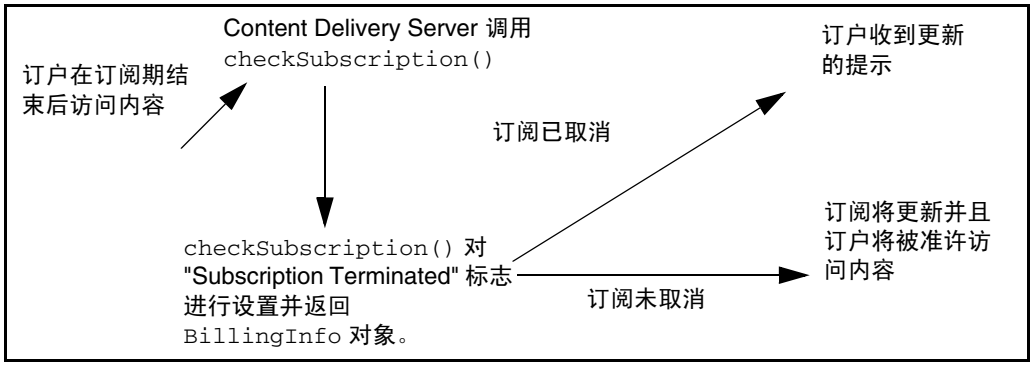


1. 当 Content Delivery Server 从设备收到下载成功的确认消息时，Content Delivery Server 将检查记帐信息，以查看是否需要将该确认消息通知记帐系统。
2. 如果需要确认， Content Delivery Server 将调用记帐适配器的 `confirm()` 方法。有关其他信息，请参见第 29 页上的第 3.2.3 节 “confirm()”。
3. 如果设备返回的不是成功确认的消息，而是错误消息，Content Delivery Server 将调用记帐适配器的 `reverse()` 方法，并传递此事务的 `BillingInfo` 对象。`reverse()` 的实现应确保订户不会为未下载的内容支付费用。有关其他信息，请参见第 31 页上的第 3.2.8 节 “reverse()”。

3.1.5 订阅验证

当订户在订阅期结束后访问内容时，订阅验证进程将启动。图 6 显示了此进程。

图 6 订阅验证进程流



1. 当订户在订阅期结束后访问内容时， Content Delivery Server 将调用 `checkSubscription()`，以查看是否已在 Content Delivery Server 外部取消了订阅。
2. `checkSubscription()` 的实现应在返回到 Content Delivery Server 的 `BillingInfo` 对象中设置 "Subscription Terminated" 标志。有关其他信息，请参见第 29 页上的第 3.2.2 节 “checkSubscription()”。
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()` 将 `BillingInfo` 对象中的 "OK" 标志设置为 false，并调用 `setReplyMessage()` 设置向订户显示的消息。

3.2 BillingManager 接口

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

BillingManager 接口位于 com.sun.content.server.billing 包中。

3.2.1 authorize()

```
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，此方法将被 Content Delivery Server 调用。要表明需要授权， BillingInfo 对象中的 “Authorize Needed”标志必须由 getBillingInfo() 进行设置。

使用此方法可以确定订户是否经过授权购买所需内容。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 提供了按 BillingConstants 类中定义的记帐模型常数进行编索的标志数组。这些常数是

- DOWNLOAD：下载费用
- SUBSCRIPTION：订阅费用
- TRIAL：提供免费试用期
- USAGE：按使用次数收费

每个标志都表明授权事务时是否应考虑记帐模型。例如，以下参数已给定：

- 记帐模型已下载
- 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，以表明无需为此事务收取下载费用。因此，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，则可以在将 BillingInfo 对象返回到 Content Delivery Server 之前调用 setIsOK()，以便将 BillingInfo 对象中的 “OK”标志设置为 true。如果不允许订户购买该内容，则将 “OK”标志设置为 false。

如果未通过 getBillingInfo() 设置 BillingInfo 对象中的 “Confirm Needed”标志，则可以通过调用 setConfirmNeeded() 来设置该标志。如果希望在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()

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

如果订户试图在订阅期结束后使用内容， Content Delivery Server 将调用此方法。使用该方法可以在订阅在 Content Delivery Server 外部终止时通知 Content Delivery Server。

此方法应通过调用 `BillingInfo` 对象的 `setSubscriptionTerminated()` 方法设置 "Subscription Terminated" 标志。要表明订阅已终止，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()

```
abstract void 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此方法。要表明需要确认， `BillingInfo` 对象中的 "Confirm Needed" 标志必须由 `getBillingInfo()` 或 `authorize()` 进行设置。

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 getBillingInfo()

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

此方法在订户请求项目的详细资料或购买项目时由 Content Delivery Server 调用。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。

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。

此方法应通过调用 `BillingInfo` 对象的 `setAuthorizeNeeded()` 方法设置 "Authorize Needed" 标志。如果希望记帐系统验证订户是否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。如果希望在内容成功下载到设备时通知记帐系统，请通过调用 `setConfirmNeeded()` 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()` 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`，订户将不能下载指定内容。如果 "Authorize Needed" 标志设置为 `true`，则 "OK" 标志应在 `authorize()` 中进行设置。

3.2.5 getBillingInfos()

```
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，此方法将由 Content Delivery Server 调用。 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。

此方法应通过调用列表中的每个 `BillingInfo` 对象的 `setAuthorizeNeeded()` 方法设置 "Authorize Needed" 标志。如果希望记帐系统验证订户是否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。如果希望在内容成功下载到设备时通知记帐系统，请通过调用 `setConfirmNeeded()` 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()` 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`，订户将不能下载指定内容。如果 "Authorize Needed" 标志设置为 `true`，则 "OK" 标志应在 `authorize()` 中进行设置。

3.2.6 getLog()

```
protected static com.sun.content.server.log.LogCategory getLog()
```

此方法返回 `LogCatetory` 对象，用于记录记帐 API 生成的错误和警告消息。使用此方法可以获得应在处理 Content Delivery Server 中的记帐事务时用于记录错误和其他信息的日志文件。

3.2.7 refund()

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

此方法将在客户服务代理使用 Vending Manager 管理控制台退还订户购买费用时由 Content Delivery Server 调用。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()

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

此方法在将内容无法下载到订户设备时由 Content Delivery Server 调用。 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，因此订户便无需为该内容付费。

3.2.9 subscribe()

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

此方法在订户开始订阅内容时由 Content Delivery Server 调用。 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。当每次订阅期结束时，记帐系统应自动向订户收取费用。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()

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

此方法在订户取消重复出现的订阅时由 Content Delivery Server 调用。 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期结束时停止自动付费。

3.3 使用记帐 API

记帐 API 的类可以在 `cdsapi.jar`（位于 `$CDS_HOME/deployment/`*`deployment-name`*`/lib/cdslib` 目录）中找到。

编译适配器时， `cdsapi.jar` 文件必须位于类路径中。

为使适配器可用于 Content Delivery Server，请执行以下操作：

1. **为适配器创建 Java 归档 (JAR) 文件，并将 JAR 文件置于** `$CDS_HOME/deployment/`***deployment-name***`/lib/external` **目录中，以便在执行过程中 Content Delivery Server 可以找到该文件。**

   Content Delivery Server 需要重新启动才能发觉新的 JAR 文件。

2. **在** `$CDS_HOME/deployment/`***deployment-name***`/conf` **目录中修改** `security.config` **文件来设置** `module.security.billingmanager` **属性。**

   将此属性设置为 `BillingManager` 的实现的类名。例如，

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

3.4 示例

代码示例 3 显示了 BillingManager 的缺省实现。

代码示例 3 BillingManager 样例实现

```
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;
import com.sun.content.server.log.BillingManagerKeys;
import com.sun.content.server.log.LogCategory;

/**
 * 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;
  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;

  /**
   * This is used to log debug, warning, and error messages to the
   * logging system.
   */
  private static final LogCategory sLog =
    LogCategory.getLog("BillingManager");

  /**
   * 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");

  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 void 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");
  }

  /**
   * 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");

  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对内容二进制进行程序校验或更改内容信息。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 (DRM) 的代码。

注一如果内容管理适配器需要在开发者提交的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程序校验或修改。有关创建工作流的信息，请参见《*Sun Java System Content Delivery Server 集成指南*》。有关配置 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 安装指南*》。

请注意，内容管理适配器处理收到的调用需要一些时间，这会延迟到订户的内容传送。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

- `ContentManager` 接口：用于实现将 Content Delivery Server 与内容管理系统进行集成的接口。 `ContentManager` 接口提供了多种方法以访问有关内容的信息。
- `ContentInfo` 类：一种类，包含内容对象信息，例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、内容大小和类型 （例如 MIDlet、ringtone 等）以及事务详细信息，例如一步或两步下载。
- `ContentException` 类：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` 类：定义访问内容元数据的 `ContentInfo` 对象使用的关键字值的类。

有关这些类的其他信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 处的 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。内容和事务的详细资料保留在 `ContentInfo` 对象中。使用此 API 可以访问和操作内容传送给订户时的内容信息。

本节介绍设备、 Content Delivery Server 及内容管理适配器之间的信息流。

- 获取内容列表
- 获取内容详细信息
- 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。

1. 订户在 Subscriber Portal 中单击“**类别**”下的内容标题。
2. 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。
3. 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。
4. Content Delivery Server 调用 `getContentInfos` 并传递列表中各项的 `ContentInfo` 对象。
5. `getContentInfos` 的实现可以更改系统所需的内容信息。此时只有内容列表中显示给订户的信息应进行修改。
6. Content Delivery Server 将该类别的内容项目列表和其他信息返回到订户设备。
7. Subscriber Portal 在页面的 “结果”部分中显示该列表。

4.1.2 获取内容详细信息

当订户在 Subscriber Portal 的 “结果”页面中单击内容项名称或“**更多详细资料**”时，将启动有关特定内容项详细信息的请求。事务步骤将在下面进行介绍。

1. 订户在 Subscriber Portal 的 “结果”页面中单击特定内容项的名称或“**更多详细资料**”。
2. 订户设备向 Content Delivery Server 发送请求以获得选定内容项目的信息。

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，例如其二进制代码和描述符 MIME 类型、其内容大小和类型 （例如 ringtone 或 MIDlet）以及事务详细信息。

4. Content Delivery Server 将有关内容项目的详细信息传递到订户设备。

4.1.3 下载内容

订户选择从设备购买内容时将启动获取内容的进程。通常，该内容必须在传送到订户之前进行程序校验。可以根据很多因素进行程序校验，例如订户电话类型 （CDMA 或 GSM）、记帐模型 （订阅或使用）、内容类型或其他特征。

1. 订户在 Subscriber Portal 中选择所需的内容项，然后单击 **“现在下载”**。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 数据库中检索二进制代码、描述符及其 MIME 类型，并将它们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程序校验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－功能匹配 （用于确定运行内容所用的设备）在发布内容时完成。内容管理适配器所做的更改不应包含这样的内容，即致使设备不能再运行内容。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。您可以对其提供的方法进行实现，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它们进行修改。

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，例如内容二进制代码和描述符 MIME 类型、内容大小和类型，以及下载内容所需的步骤数。

BillingInfo 对象包含事务详细资料，例如记帐模型。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 “记帐 API”。

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

4.2.1 getContentInfo()

```
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 对象包含关于初始内容项的信息，例如内容二进制代码和描述符 MIME 类型、估计的内容大小和类型 （例如 ringtone 或 MIDlet）以及下载该内容所需的步骤数。

4.2.2 getContentInfos()

```
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。此方法只应该更改内容列表中显示的信息。

4.2.3 getContentDescriptor()

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

此方法在订户启动请求将内容描述符下载到设备上时由 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和信息 （包括大小），并将其返回到 Content Delivery Server，以便传递给订户。

4.2.4 getContentBinary()

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

此方法在订户启动请求将内容二进制代码下载到设备上时由 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。要传送给订户经过程序校验的内容二进制代码必须返回到 Content Delivery Server。

4.3 使用内容管理 API

内容管理 API 的类可以在 `cdsapi.jar`（位于 `$CDS_HOME/deployment/`*`deployment-name`*`/lib/cdslib` 目录）中找到。

编译适配器时， `cdsapi.jar` 文件和 `foundation.jar` 文件必须位于类路径中。

要使适配器可用于 Content Delivery Server：

1. **为适配器创建 Java 归档 (JAR) 文件，并将 JAR 文件置于** `$CDS_HOME/deployment/`***deployment-name***`/lib/external` **目录中，以便在执行过程中 Content Delivery Server 可以找到该文件。**

   Content Delivery Server 需要重新启动才能发觉新的 JAR 文件。

2. **修改** `$CDS_HOME/deployment/`***deployment-name***`/conf` **目录中的** `security.config` **文件，以添加** `module.security.contentmanager` **属性和设置** `module.security.contentmanager.enabled` **属性。**

   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。在现有的那一个属性后面再添加一个 `module.security.contentmanager` 属性，并将此属性设置为您所实现的 `ContentManager` 接口的全限定包和类名。将 `module.security.contentmanager.enabled` 设置为 **`true`** 以表明适配器可用，并且该属性应由 Content Delivery Server 调用。例如，

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

4.4 示例

代码示例 4 显示了 ContentManager 接口的样例实现。

代码示例 4 ContentManager 样例实现

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

public class ContentManagerImpl implements ContentManager
{
  public ContentInfo getContentInfo(
    ContentInfo inContentInfo,
    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 的内容验证 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，您可以扩展该类来创建自己的内容验证适配器。请实现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，您可以扩展该类来创建自己的 `ValidationContent` 对象。 ValidationContent 对象包含标识被提交内容的信息。

有关这些类的其他信息，请参见 `$CDS_HOME/javadoc/validation/index.html` 处的 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二进制内容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 (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 集成指南*》。

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二进制内容。 `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 类处理元数据和二进制内容，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()

```
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 调用此方法。此时传入前一步的 ValidationContent 对象以及工作流步骤中指定的参数。您必须知道在工作流上一步中创建的 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。

在实现此方法的过程中，必须处理传入的属性，并创建要返回的 ValidationContent 对象。根据需要处理元数据和二进制内容。例如，如果希望对代码进行混淆处理，那么要调用所需的混淆器来转换二进制内容，并在 ValidationContent 对象中返回新的二进制内容。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，那么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，那么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/SumbissionVerifierAdapters.xml 文件中，设置适配器的属性文件名，如第 47 页上的 “使用内容验证 API”中所述。

5.2.2 returns()

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

此方法由 Content Delivery Server 调用，验证适配器能否处理要传入的 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，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 48 页上的 “示例”。

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，那么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

内容验证 API 的类位于软件包 com.sun.content.server.validation.adapter 中。此软件包包含在 validation.jar 文件中，该文件位于以下某个位置：

- 对于 Sun Java System Application Server，该文件位于 $CDS_HOME/deployment/*deployment-name*/sun/domains/*server-domain*/*server-name*/applications/j2ee-modules/CDSDeveloperPortal_1/WEB-INF/lib。
- 对于 WebLogic Server，该文件位于 $CDS_HOME/deployment/*deployment-name*/weblogic/domains/*server-domain*/applications/developerportal/WEB-INF/lib。

其中， *deployment-name* 是部署 Developer Portal 时指定的名称， *server-domain* 是 app.server.domain 属性的配置文件中指定的值， *server-name* 是 app.server.name 属性的配置文件中指定的值。

编译适配器时， validation.jar 文件必须位于类路径中。

为使适配器可用于 Content Delivery Server，请执行以下操作：

1. **为适配器创建 Java 归档 (JAR) 文件，并将 JAR 文件置于** `CDS_HOME/deployment/`***deployment-name***`/lib/external` **目录中，以便在执行过程中 Content Delivery Server 可以找到该文件。**

   Content Delivery Server 需要重新启动才能发觉新的 JAR 文件。

2. **修改** `$CDS_HOME/deployment/`***deployment-name***`/conf` **目录中的** `SubmissionVerifierAdapters.xml` **文件，在其中为所创建的适配器加入一条语句。**

   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，那么请将以下语句添加到文件中：

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

3. **修改** `$CDS_HOME/deployment/`***deployment-name***`/conf` **目录中的** `SubmissionVerifierWorkflows.xml` **文件，加入一个步骤执行所创建的适配器。**

   有关步骤元素的适配器特性的值，请指定为上一步中添加到 `SubmissionVerifierAdapters.xml` 文件中的适配器元素 id 特性提供的值。有关创建工作流的信息，请参见 《*Sun Java System Content Delivery Server 集成指南*》。

5.5 示例

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。

代码示例 5 ValidationAdapter 样例实现

```
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
    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

用户配置文件 API

本章介绍了 Sun Java System Content Delivery Server 用户管理 API。使用此 API 可以添加、删除、检索、更新、启用或禁用系统中的用户。

`UserManager` 类用于控制个人用户帐户的创建和状态，并且必须由 `UserManager` 类实现来实现。

`User` 接口用于管理每个用户的特定特性。

`UserDeviceManager` 接口提供有关订户正在使用的设备的信息，并可以用来防止订户只付一次费而将内容下载到多个设备上的情况。

有关本节中未介绍的类或方法的信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 中的用户配置文件 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。

6.1.1 doAccountExists()

如果用户帐户已经存在于给定用户 ID 的永久性存储中，将返回 `true`。

```
abstract boolean doAccountExists(String uId)
```

6.1.2 doAddUser()

创建新用户。

```
protected abstract boolean doAddUser(User u)
```

6.1.3 doDisableUser()

禁用用户帐户。用于锁定用户帐户。

```
protected abstract void doDisableUser(String uId)
```

6.1.4 doEnableUser()

启用用户帐户。

```
protected abstract void doEnableUser(String uId)
```

6.1.5 doGetAllLikeInOrder()

按指定顺序返回请求对象的迭代程序。

```
protected abstract void doAllLikeInOrder(String[] columns, String[]
    values, String[] orders, boolean isDescending, int pageNum, int
    recPerPage, String role)
```

6.1.6 doGetAllUsers()

根据用户角色返回永久存储中所有用户的迭代程序。

```
protected abstract Iterator doGetAllUsers(String role)
```

6.1.7 doGetAllUsersContainingFirstName()

根据用户角色返回名字中包含指定子字符串的所有用户的迭代程序。

```
protected abstract Iterator doGetAllUsersContainingFirstName(String
    nameLike, String role)
```

6.1.8 doGetAllUsersContainingId()

根据用户角色返回用户 ID 中包含指定子字符串的所有用户的迭代程序。

```
protected abstract Iterator doGetAllUsersContainingId(String id,
    String role)
```

6.1.9 doGetAllUsersContainingLastName()

根据用户角色返回姓氏中包含指定子字符串的所有用户的迭代程序。

```
protected abstract Iterator doGetAllUsersContainingLastName(String
    lastName, String role)
```

6.1.10 doGetAllUsersContainingName()

根据用户角色返回姓氏或名字中包含指定子字符串的所有用户的迭代程序。

```
protected abstract Iterator doGetAllUsersContainingName(String
    name, String role)
```

6.1.11 doGetAnonymousUser()

为匿名访问系统返回一个来宾用户。

```
protected abstract User doGetAnonymousUser()
```

6.1.12 doGetFieldName()

根据用户角色获取对应于某个字段常量的外部数据库中的字段名称。

```
protected abstract String doGetFieldName(int fieldContsant, String
    role)
```

6.1.13 doGetUser()

返回与给定用户 ID 关联的用户对象。

```
protected abstract User doGetUser(String uId)
```

6.1.14 doGetUserByMobileId()

返回与给定用户移动 ID 关联的用户对象。

```
protected abstract User doGetUserByMobileId(String inMobileId)
```

6.1.15 doGetUserByUniqueDeviceId()

返回与给定用户唯一设备 ID 关联的用户对象。

```
protected abstract User doGetUserByUniqueDeviceId(String
    inUniqueDeviceId)
```

6.1.16 doGetUserInstance()

返回 `addUser` 方法中传递的用户类实现的实例。

```
protected abstract User doGetUserInstance()
```

6.1.17 doIsActive()

如果与 `userId` 关联的用户帐户处于活动状态，则返回 `true`。如果帐户处于非活动状态，则返回 `false`。

```
protected abstract boolean doIsActive(String uId)
```

6.1.18 doIsAuthenticated()

如果允许通过提供的用户 ID 和密码进行访问，则返回 `true`。

```
protected abstract boolean doIsAuthenticated(String userId, String
    inPassword)
```

6.1.19 doRemoveUser()

将用户从永久存储中删除。如果成功，则返回 `true`。

```
protected abstract boolean doRemoveUser(String uId)
```

6.1.20 doUpdateUser()

更新现有的用户帐户。

```
protected abstract boolean doUpdateUser(User u)
```

6.2 User 接口

User 接口用于定义获取、设置或删除用户特性的通用方法。

6.2.1 getActivateDate()

返回激活用户帐户的日期。

```
public Date getActivateDate()
```

6.2.2 getAttribute()

返回与特定用户关联的特性值。

```
public Object getAttribute ( String name )
```

6.2.3 getAttribute()

返回与特定用户关联的特定特性的值或默认值 （如果未找到属性）。

```
public Object getAttribute ( String name, Object defaultValue )
```

6.2.4 getAttributes()

返回包含与特定用户关联的所有特性的散列表。

```
public Hashtable getAttributes()
```

6.2.5 getCity()

返回城市名称。

```
public String getCity()
```

6.2.6 getCountry()

返回国家 / 地区名称。

```
public String getCountry()
```

6.2.7 getCreateDate()

返回用户帐户的创建日期。

```
public java.util.Date getCreateDate()
```

6.2.8 getDeActivateDate()

返回用户帐户的激活日期。

```
public Date getDeActivateDate()
```

6.2.9 getEmail()

返回用户的电子邮件 ID。

```
public String getEmail()
```

6.2.10 getFirstName()

返回用户的名字。

```
public String getFirstName()
```

6.2.11 getGender()

返回用户的性别。

```
public String getGender()
```

6.2.12 getLastLogin()

返回用户上次登录时间。

```
public java.util.Date getLastLogin()
```

6.2.13 getLastName()

返回用户的姓氏。

```
public String getLastName()
```

6.2.14 getLoginId()

返回用户的唯一登录 ID。

```
public String getLoginId()
```

6.2.15 getMiddleName()

返回用户的中间名。

```
public String getMiddleName()
```

6.2.16 getMobileId()

返回用户的移动 ID。

```
public String getMobileId()
```

6.2.17 getPassword()

返回用户的唯一密码。

```
public String getPassword()
```

6.2.18 getPhone()

返回用户的联系电话号码。

```
public String getPhone()
```

6.2.19 getPostalCode()

返回用户帐户的邮政编码。

```
public String getPostalCode()
```

6.2.20 getSalutation()

返回用户的称呼。

```
public String getSalutation()
```

6.2.21 getState()

返回省 / 市 / 自治区名称。

```
public String getState()
```

6.2.22 getStreet1()

返回用户街道地址的第一行。

```
public String getStreet1()
```

6.2.23 getStreet2()

返回用户街道地址的第二行。

```
public String getStreet2()
```

6.2.24 getUniqueDeviceId()

返回用户的唯一设备 ID。

```
public String getUniqueDeviceId()
```

6.2.25 hasLoggedIn()

如果用户已经登录到系统，则返回 `true`。

```
public boolean hasLoggedIn()
```

6.2.26 isConfirmed()

如果用户帐户已确认或验证，则返回 `true`。

```
public boolean isConfirmed()
```

6.2.27 isEnabled()

如果用户帐户当前已启用，则返回 `true`；如果已禁用，则返回 `false`。

```
public boolean isEnabled()
```

6.2.28 isPrepay()

如果用户帐户已预付，则返回 `true`。

```
public boolean isPrepay()
```

6.2.29 setActivateDate()

设置用户帐户的激活日期。

```
public void setActivateDate(Date aDate)
```

6.2.30 setAttribute()

设置与特定用户关联的特性。

```
public void setAttribute ( String name, Object value )
```

6.2.31 setAttributes()

设置与特定用户关联的特性的列表。

```
public void setAttributes(Hashtable stuff)
```

6.2.32 setCity()

设置城市名称。

```
public void setCity(String city)
```

6.2.33 setCountry()

设置国家 / 地区名称。

```
public void setCountry(String country)
```

6.2.34 setCreateDate()

设置用户帐户的创建日期。

```
public void setCreateDate(java.util.Date date)
```

6.2.35 setDeActivateDate()

设置用户帐户取消激活的日期。

```
public void setDeActivateDate(Date daDate)
```

6.2.36 setEmail()

设置用户的电子邮件 ID。

```
public void setEmail(String address)
```

6.2.37 setFirstName()

设置用户的名字。

```
public void setFirstName(String firstName)
```

6.2.38 setGender()

设置用户的性别。

```
public void setGender(String gender)
```

6.2.39 setHasLoggedIn()

设置用来表明用户已登录到系统的标志。

```
public void setHasLoggedIn(boolean value)
```

6.2.40 setIsEnabled()

将帐户状态设置为 “已启用”。

```
public void setIsEnabled(boolean value)
```

6.2.41 setIsPrepay()

将帐户设置为 “已预付”。

```
public void setIsPrepay(boolean value)
```

6.2.42 setLastName()

设置用户的姓氏。

```
public void setLastName(String lastName)
```

6.2.43 setLoginId()

设置用户的唯一登录 ID。

```
public void setLoginId(String loginName)
```

6.2.44 setMiddleName()

设置用户的中间名。

```
public void setMiddleName(String name)
```

6.2.45 setMobileId()

设置用户的移动 ID。

```
public void setMobileId(String mobileId)
```

6.2.46 setPassword()

设置用户的密码。

```
public void setPassword(String pass)
```

6.2.47 setPhone()

设置用户的联系电话号码。

```
public void setPhone(String phone)
```

6.2.48 setPostalCode()

设置用户地址的邮政编码。

```
public void setPostalCode(String zip)
```

6.2.49 setSalutation()

设置用户称呼（例如先生或女士）。

```
public void setSalutation(String salutation)
```

6.2.50 setState()

设置用户地址的省 / 市 / 自治区。

```
public void setState(String state)
```

6.2.51 setStreet1()

设置用户街道地址的第一行。

```
public void setStreet1(String st1)
```

6.2.52 setStreet2()

设置用户街道地址的第二行。

```
public void setStreet2(String st2)
```

6.2.53 setUniqueDeviceId()

设置用户的唯一设备 ID。 .

```
public void setUniqueDeviceId(String uniqueId)
```

6.2.54 updateLastLogin()

使用上次登录时间更新时间标记。

```
public void updateLastLogin() throws Exception
```

6.3 UserDeviceManager 接口

6.3.1 getUniqueDeviceID()

获取用于设备的唯一设备 ID，例如 MSISDN 提供的 ESN （电子序列号）。

```
public String getUniqueDeviceID(String inMSISDN) throws
    UserDeviceException
```

6.4 使用用户配置文件 API

用户配置文件 API 的类可以在 `cdsapi.jar` （位于 `$CDS_HOME/deployment/`*deployment-name*`/lib/cdslib` 目录）中找到。

编译适配器时， `cdsapi.jar` 文件必须位于类路径中。

为使适配器可用于 Content Delivery Server，请执行以下操作：

1. **为适配器创建 Java 归档 (JAR) 文件，并将 JAR 文件置于** `CDS_HOME/deployment/`***deployment-name***`/lib/external` **目录中，以便在执行过程中 Content Delivery Server 可以找到该文件。**

   Content Delivery Server 需要重新启动才能发觉新的 JAR 文件。

2. **在** `$CDS_HOME/deployment/`***deployment-name***`/conf` **目录中修改** `security.config` **文件来设置** `module.security.usermanagerfactory` **和** `module.security.subscriber.usermanager` **属性。**

   将 `module.security.subscriber.usermanager` 设置为 `UserManager` 类的实现的名称。例如，

   ```
   module.security.usermanagerfactory=com.sun.content.server.service.
       security.UserManagerFactory
   module.security.subscriber.usermanager=com.sun.content.server.serv
       er.security.user.UserImpl
   ```

6.5 示例

下例是 MyUserMgr 的类定义。该示例介绍了如何继承 UserManager 类以管理用户配置文件系统并将其集成到 Content Delivery Server 用户配置文件框架中。以下样例类提供了一个示例，即如何使用用户配置文件 API 来管理用户配置文件系统并将其集成到 Content Delivery Server 用户配置文件框架中。

6.5.1 支持文件

用户配置文件 API 的样例实现使用特性和一个支持类。以下代码显示了所用的各个特性。

代码示例 6 operatorproxy.properties

```
# This is a sample of typical Operator Integration properties file
# for external user manager client/server connection settings.
#

# SAMPLE OPERATOR CONFIGURATIONS
#
externalserveraddress=localhost
externalserverport=7779
cdsclientlogin=cds14sun
cdsclientpasswd=cds4sun1
```

SampleExternalProxy 类用作 Content Delivery Server 与外部数据库之间的接口。

代码示例 7 SampleExternalProxy.java

```
package com.sun.content.server.operator.security.adaptor;

import java.util.*;
import com.sun.content.server.service.security.*;
import com.sun.content.server.service.security.util.*;

// import external required packages to connect to the directory
   service

public class SampleExternalProxy
{
     // Create a Client facace for serach
    private ExternalClientObject externalUserMgr;

    // Create an instance of External Directorory server Client Proxy
    public SampleExternalProxy() throws Exception {

// User the external package to instanciate the Proxu
// You need to use the client/server setting in the
    // operatorproxy.properties
// The following assume you implemented a class
   OperatorProxyProperties
    // to read the configuration values.

String hostname = OperatorProxyProperties.EXTERNAL_SERVER_ADDRESS;
String port = OperatorProxyProperties.EXTERNAL_SERVER_PORT;
String clientLoginId = OperatorProxyProperties.CLIENT_LOGIN;
    String clientPassword = OperatorProxyProperties.CLIENT_PASSWRD;

    // instan
    try {
externalUserMgr = new ExternalClientObject(hostname, port,
            clientLoginId, clientPassword);
    } catch (Exception expt) {
    // process the exception
         throw expt;
    }
}

    public boolean searchUser(String userName) {

        boolean found = false;
// Use the external User manager and search function for the given
   string
    // name
 // Typically the call will look like
 // found = externalUserMgr.searchUser(userName);
 // You may need to catch potential exception and display the
   appropriate
    // message
```

```
        return found;
    }

    public SampleUserImpl createUserFromExternal(String loginId)
            throw Exception {

System.out.println("DEBUG:SampleExternal Proxy ---
   createUserFromExternal ");
System.out.println(" Creating a SampleUserImpl from External
   Directory ....");
System.out.println(" Reference Login Id = "+loginId);

        try {

// Assuming the external client has been created and connection is
      // established
// The following call typically search for the UserName and return an
      // External UserProfile
aUserProfileData aUPD =
   externalUserMgr.getUserProfileData(loginId);
String password = aUPD.getCredential();

            // Create basic User information (firstname,
            // middlename, lastname,
            // address, etc.)
            String firstName = aUPD.getFirstName();
            String lastName = aUPD.getLastName();
            String middleName = aUPD.getMiddleInitial();
            String gender = aUPD.getGender();
            String salutation = aUPD.getOccupation();
            String street1 = aUPD.getStreet();
            String street2 = aUPD.getStreetNumber();
            String city = aUPD.getCity();
            String state = aUPD.getState();
            String postalcode = aUPD.getZipCode();
            String country = aUPD.getCountry();
            String phone = aUPD.getFixedPhone();

            // Creating email information
            String email = aUPD.getMailAddress();

            // Creating msisdn in this case Unique Device Id ()
            String uniqueDeviceId = aUPD.getMsisdn();

            // Creating Status data:User Enabled/Desabled and User
   is
            // Prepay/Non-Prepay are valued 1/0

            boolean enabled = false;
            if (aUPD.getStatus().equals("1"))
                enabled = true;
```

```
            boolean prepay = false;
            if (aUPD.getPrepayType().equals("1"))
                prepay = true;

            // Create activation and deactivation dates if provided
            Date activatedate = new Date();
            Date deactivatedate = null;

// Create and return a Sample User Implementation
            return new SampleUserImpl(
                loginId,
                password,
                firstName,
                lastName,
                middleName,
                gender,
                street1,
                street2,
                city,
                state,
                postalcode,
                country,
                email,
                phone,
                activatedate,
                deactivatedate,
                salutation,
                enabled,
    uniqueDeviceId,
                prepay);

        } catch (Exception ex) {
            // procecss exception
            throw ex;
        }

        return null;
    }
}
```

6.5.2 SampleUserImpl.java

以下代码是用户接口的样例实现。此接口包括 Content Delivery Server 用于用户配置文件的字段。对于特定实现可能还需要添加其他字段。如果需要添加其他字段，请添加所需的方法以获得和设置这些值。

代码示例 8 SampleUserImpl.java

```
package com.sun.content.server.operator.security.adaptor;
import com.sun.content.server.service.security.User;
import java.util.Date;
import java.util.Hashtable;

public class SampleUserImpl implements User
{

    private Hashtable fInfo;

    public SampleUserImpl() {

        fInfo = new Hashtable();

        setLoginId("guest");
        setFirstName("guest");
        setLastName("guest");
        setPassword ("guest");
        setCreateDate(new Date());
        setActivateDate(new Date());
        fInfo.put("enabled", String.valueOf(true));
        setMiddleName("guest");
        setEmail("guest@email.com");
        setUniqueDeviceId("1231231233");
    }

    public SampleUserImpl(String uid, String pwd, String fname,
   String lname,String mname,String gender,
       String street1, String street2,String city, String
   state,String postalcode,
       String country,String email,String phone,Date actdate,Date
   deactdate,
       String salutation, boolean enabled, String uniqueDeviceId,
   boolean isprepay) {

        fInfo = new Hashtable();

        setLoginId(uid);
        setPassword(pwd);
        setFirstName(fname);
        setLastName(lname);
        setMiddleName(mname);
        setGender(gender);
        setStreet1(street1);
        setStreet2(street2);
        setCity(city);
        setState(state);
        setPostalCode(postalcode);
        setCountry(country);
        setEmail(email);
```

```
        setPhone(phone);
        setFixedPhone(phone);

        // It is safe to check the activation date and use current
    date if null
        if ( actdate == null ) {
            setCreateDate(new Date());
            setActivateDate(new Date());
        }
        else {
            setCreateDate(actdate);
            setActivateDate(actdate);
        }

   setDeActivateDate(deactdate);
        setSalutation(salutation);
        setIsEnabled(enabled);
        setUniqueDeviceId(uniqueDeviceId);
        setIsPrepay(isprepay);
    }

    public SampleUserImpl(User inUser) {

        fInfo = new Hashtable();

        setLoginId(inUser.getLoginId());
        setFirstName(inUser.getFirstName());
        setLastName(inUser.getLastName());
        setMiddleName(inUser.getMiddleName());
        setGender(inUser.getGender());
        setStreet1(inUser.getStreet1());
        setStreet2(inUser.getStreet2());
        setCity(inUser.getCity());
        setState(inUser.getState());
        setPostalCode(inUser.getPostalCode());
        setCountry(inUser.getCountry());
        setEmail(inUser.getEmail());
        setPhone(inUser.getPhone());
        setSalutation(inUser.getSalutation());
        setIsEnabled(inUser.isEnabled());
        setPassword(inUser.getPassword());
        setUniqueDeviceId(inUser.getUniqueDeviceId());
    }


// It is sometime useful to have the corresponding external user data
    as a
  // hashtable
    public Hashtable getExternalUserData()
    {
    // Create a hash table
Hashtable externalData = new Hashtable();
```

```
// Get all the external data from fInfo and update the external data
// externalData = parserUser(fInfo);
        return externalData;
    }

    public Date getLastLogin() {
        /**@todo:Implement this
   com.sun.content.server.service.security.User method*/
        throw new java.lang.UnsupportedOperationException("Method
   getLastLogin() not yet implemented.");
  }

    public Object getAttribute(String param1) {
    /**@todo:Implement this
   com.sun.content.server.service.security.User method*/
    throw new java.lang.UnsupportedOperationException("Method
   getAttribute() not yet implemented.");
    }

    public Object getAttribute(String param1, Object parm2) {
        /**@todo:Implement this
   com.sun.content.server.service.security.User method*/
        throw new java.lang.UnsupportedOperationException("Method
   getAttribute() not yet implemented.");
    }

    public Hashtable getAttributes() {
    /**@todo:Implement this
   com.sun.content.server.service.security.User method*/
    throw new java.lang.UnsupportedOperationException("Method
   getAttributes() not yet implemented.");
    }

    public void setHasLoggedIn(boolean param1) {
/**@todo:Implement this
   com.sun.content.server.service.security.User method*/
throw new java.lang.UnsupportedOperationException("Method
   setAttribute() not yet implemented.");
    }
public boolean hasLoggedIn() {
/**@todo:Implement this
   com.sun.content.server.service.security.User method*/
throw new java.lang.UnsupportedOperationException("Method
   isConfirmed() not yet implemented.");
    }

    public void setAttribute(String param1, Object parm2) {
        /**@todo:Implement this
   com.sun.content.server.service.security.User method*/
        throw new java.lang.UnsupportedOperationException("Method
   setAttribute() not yet implemented.");
    }
```

```
 public void setAttributes(Hashtable param1) {
     /**@todo:Implement this
com.sun.content.server.service.security.User method*/
     throw new java.lang.UnsupportedOperationException("Method
setAttributes() not yet implemented.");
 }


 public String getLoginId() {
     return (String)fInfo.get("loginId");
 }
 public void setLoginId(String param1) {
     if (param1 != null)
     fInfo.put("loginId", param1);
 }

 public String getFirstName() {
     return (String)fInfo.get("firstName");
 }
 public void setFirstName(String param1) {
     if (param1 != null)
     fInfo.put("firstName", param1);
 }

 public String getLastName() {
     return (String)fInfo.get("lastName");
 }
 public void setLastName(String param1) {
     if (param1 != null)
         fInfo.put("lastName", param1);
 }

 public Date getCreateDate() {
         return (Date)fInfo.get("createDate");
 }
 public void setCreateDate(Date param1) {
     if (param1 != null)
         fInfo.put("createDate", param1);
 }

 public String getEmail() {
         return (String)fInfo.get("email");
 }
 public void setEmail(String param1) {
     if (param1 != null)
         fInfo.put("email", param1);
 }

 public boolean isConfirmed() {
     /**@todo:Implement this
com.sun.content.server.service.security.User method*/
```

```
        throw new java.lang.UnsupportedOperationException("Method
    isConfirmed() not yet implemented.");
    }

    public void updateLastLogin() throws java.lang.Exception {
        /**@todo:Implement this
    com.sun.content.server.service.security.User method*/
        throw new java.lang.UnsupportedOperationException("Method
    updateLastLogin() not yet implemented.");
    }

    public String getMiddleName() {
        return (String)fInfo.get("middleName");
    }
    public void setMiddleName(String param1) {
        if (param1 != null)
            fInfo.put("middleName", param1);
    }

    public String getGender() {
        return (String)fInfo.get("gender");
    }
    public void setGender(String param1) {
        if (param1 != null)
            fInfo.put("gender", param1);
    }

    public String getStreet1() {
        return (String)fInfo.get("street1");
    }
    public void setStreet1(String param1) {
        if (param1 != null)
            fInfo.put("street1", param1);
    }

public String getStreet2() {
    return (String)fInfo.get("street2");
}
public void setStreet2(String param1) {
    if (param1 != null)
        fInfo.put("street2", param1);
}

    public String getPostalCode() {
        return (String)fInfo.get("postalCode");
    }
    public void setPostalCode(String param1) {
        if (param1 != null)
            fInfo.put("postalCode", param1);
    }
```

```
    public String getCity() {
        return (String)fInfo.get("city");
    }
    public void setCity(String param1) {
    if (param1 != null)
        fInfo.put("city", param1);
    }


    public String getState() {
        return (String)fInfo.get("state");
    }
    public void setState(String param1) {
        if (param1 != null)
            fInfo.put("state", param1);
    }

public String getCountry() {
        return (String)fInfo.get("country");
}
public void setCountry(String param1) {
if (param1 != null)
    fInfo.put("country", param1);
}

    public String getPhone() {
        return (String)fInfo.get("phone");
    }
    public void setPhone(String param1) {
        if (param1 != null)
            fInfo.put("phone", param1);
    }

    public Date getActivateDate() {
        return (Date)fInfo.get("activateDate");
    }
    public void setActivateDate(Date param1) {
        if (param1 != null)
            fInfo.put("activateDate", param1);
    }

    public Date getDeActivateDate() {
        return (Date)fInfo.get("deactivateDate");
    }
    public void setDeActivateDate(Date param1) {
        if (param1 != null)
            fInfo.put("deactivateDate", param1);
    }

public String getSalutation() {
    return (String)fInfo.get("salutation");
}
```

```
public void setSalutation(String param1) {
if (param1 != null)
    fInfo.put("salutation", param1);
}

    public boolean isEnabled() {
        return
    Boolean.valueOf((String)fInfo.get("enabled")).booleanValue();
    }
    public void setIsEnabled(boolean param1) {
        fInfo.put("enabled", String.valueOf(param1));
    }

    public String getPassword() {
        return (String)fInfo.get("password");
    }
    public void setPassword(String param1) {
        if (param1 != null)
            fInfo.put("password", param1);
    }

   public String getUniqueDeviceId(){
            return (String)fInfo.get("UniqueDeviceId");
   }
   public void setUniqueDeviceId(String uniqueId){
            if (uniqueId != null)
                fInfo.put("UniqueDeviceId", uniqueId);
   }

public String getMobileId(){
        return (String)fInfo.get("MobileId");
}
public void setMobileId(String mobileId) {
        if (mobileId != null)
            fInfo.put("MobileId", mobileId);
}

    public boolean isPrepay() {
Boolean isprepay = (Boolean)fInfo.get("isPrepay");
return isprepay.booleanValue();
    }
public void setIsPrepay(boolean param1) {
fInfo.put("isPrepay", new Boolean(param1));
}

}
```

6.5.3 SampleUserManagerImpl.java

以下代码是 UserManager 类的样例实现。

代码示例 9 SampleUserManagerImpl.java

```
package com.sun.content.server.operator.security.adaptor;

import java.util.*;

//import Content Delivery Server libraries
import com.sun.content.server.service.security.*;
import com.sun.content.server.service.security.util.*;

//import here operator required packages
// .......

public class SampleUserManagerImpl {

private SampleExternalProxy proxy;

public SampleUserManagerImpl() throws UserProfileResourceException
    {

try {
init();
} catch (Exception ex) {
throw new
    com.sun.content.server.service.security.util.UserProfileResourc
    eException( "Failed to instantiate SampleUserManagerImpl ", ex);
}
}

// This method will create an External Directory Server Proxy and
  // initiatlize this User Manager
private void init() throws UserProfileResourceException
{
System.out.println("Initializing External UserManager .... ");

try {
proxy = new SampleExternalProxy();
} catch (Exception ex) {
System.out.println("Fatal Error " + ex.toString());
throw new
    com.sun.content.server.service.security.util.UserProfileResourc
    eException(ex.toString());
}
}

protected boolean doIsAuthenticated(String inUserId, String
    inPassword)
throws UserProfileResourceException {

System.out.println("SampleUserManagerImpl.doIsAuthenticated --- ");

boolean isauthenticated = false;
```

```
try {

User aUser = proxy.createUserFromExternal(inUserId);
isauthenticated = ( (aUser.getLoginId()==inUserId) &&
    (aUser.getPassword()==inPassword));
if (isauthenticated)
System.out.println("SampleUserManagerImpl.doIsAuthenticated:- User
    "+inUserId+" is successfully authenticated.");
else
System.out.println("SampleUserManagerImpl.doIsAuthenticated:- User
    "+inUserId+" :unable to authenticate (wrong login and
    password)");
} catch (Exception ex) {
isauthenticated = false;
System.out.println("SampleUserManagerImpl.doIsAuthenticated:- User
    "+inUserId+" can not authenticate (user not found)");
throw new
    com.sun.content.server.service.security.util.UserProfileResourc
    eException(ex.toString());
}
return isauthenticated;

}

protected boolean doAccountExists(String userId) throws
    UserProfileResourceException {

System.out.println("SampleUserManagerImpl.doAccountExists --- ");

// Use the External Proxy search method to check if the account exist
return proxy.searchUser(userId);
}


  protected boolean doAddUser(User user) throws
    UserProfileResourceException{

boolean updated = false;

//This is not an allowed operation.If this is not the case add the
    code to
     // implement it
System.out.println("SampleUserManagerImpl.doAddUser - This
    operation is not implemented !");

return updated;

  }

protected void doDisableUser(String userId) throws
    UserProfileResourceException {
```

```
//This is not an allowed operation.If this is not the case add the
   code to
    // implement it
System.out.println("SampleUserManagerImpl.doDisableUser - This
   operation is not implemented !");

}

protected void doEnableUser(String userId) throws
   UserProfileResourceException {

//This is not an allowed operation.If this is not the case add the
   code to
    // implement it
System.out.println("SampleUserManagerImpl.doEnableUser - This
   operation is not implemented !");

}

protected User doGetAnonymousUser() throws
   UserProfileResourceException {
return new SampleUserImpl();
}

protected User doGetUser(String userId) throws
   UserProfileResourceException {

User aUser = null;
try {
aUser = proxy.createUserFromExternal(userId);
if (aUser != null)

   System.out.println("SampleUserManagerImpl.doGetUserByUniqueDevi
   ceId:- User with "+userId+" is found.");
else

   System.out.println("SampleUserManagerImpl.doGetUserByUniqueDevi
   ceId:- User with "+userId+" not found");
} catch (Exception ex) {

   System.out.println("SampleUserManagerImpl.doGetUserByUniqueDevi
   ceId:- User "+userId+" not found)");
throw new
   com.sun.content.server.service.security.util.UserProfileResourc
   eException(ex);
}
return aUser;

}

/* This method can be implemented the same way as doGetUser.Instead
```

```
    of
    * using the createUserFromExternal(loginId) you can extend the
    * SampleExternalProxy to implement a specific
    * createUserFromExternalUsingDeviceId(uniqueDeviceId).
* For now we will assume that the string loginId can be replaced by
    a search
    * key ID and it will return the matching profile.
 */
protected User doGetUserByUniqueDeviceId(String inUniqueDeviceId)
    throws UserProfileResourceException {
return doGetUser(inUniqueDeviceId);
}

 // Sample implementation as doGetUser
 protected User doGetUserByMobileId(String inMobileId) throws
    UserProfileResourceException {
 return doGetUser(inMobileId);
 }

protected Iterator doGetAllUsers(String role) throws
    UserProfileResourceException {

System.out.println("SampleUserManagerImpl.doGetAllUsers - This
    operation is not implemented !");
ArrayList users = new ArrayList();
return users.iterator();
}

protected Iterator doGetAllUsersContainingLastName(String lastName,
    String role) throws UserProfileResourceException {


    System.out.println("SampleUserManagerImpl.doGetAllUsersContaini
    ngLastName - This operation is not implemented !");
ArrayList users = new ArrayList();
return users.iterator();
}

protected Iterator doGetAllUsersContainingFirstName(String
    firstName, String role) throws UserProfileResourceException {


    System.out.println("SampleUserManagerImpl.doGetAllUsersContaini
    ngFirstName - This operation is not implemented !");
ArrayList users = new ArrayList();
return users.iterator();
}

protected Iterator doGetAllUsersContainingName(String name, String
    role)
throws UserProfileResourceException {
```

```
    System.out.println("SampleUserManagerImpl.doGetAllUsersContaini
    ngName - This operation is not implemented !");
ArrayList users = new ArrayList();
return users.iterator();
}

protected Iterator doGetAllUsersContainingId(String userId, String
    role)
throws UserProfileResourceException {


    System.out.println("SampleUserManagerImpl.doGetAllUsersContaini
    ngId - This operation is not implemented !");
ArrayList users = new ArrayList();
return users.iterator();
}

protected Iterator doGetAllLikeInOrder(String[] columns, String[]
    values,
String[] orders, boolean isDescending, int pageNum, int recPerPage,
    String role)
throws UserProfileResourceException {

System.out.println("SampleUserManagerImpl.doGetAllLikeInOrder -
    This operation is not implemented !");
ArrayList users = new ArrayList();
return users.iterator();
}

protected String doGetFieldName(int fieldContant, String role)
throws UserProfileResourceException {

System.out.println("SampleUserManagerImpl.doGetFieldName - This
    operation is not implemented !");
return null;
}

protected boolean doIsActive(String userId) throws
    UserProfileResourceException {

System.out.println("SampleUserManagerImpl.doIsActive - This
    operation is not implemented !");
return false;
}

protected boolean doRemoveUser(String userId) throws
    UserProfileResourceException {

System.out.println("SampleUserManagerImpl.doRemoveUser - This
    operation is not implemented !");
return false;
```

```
}

protected boolean doUpdateUser(User user) throws
   UserProfileResourceException {

System.out.println("SampleUserManagerImpl.doUpdateUser - This
   operation is not implemented !");
return false;
}

protected User doGetUserInstance() throws
   UserProfileResourceException {

System.out.println("SampleUserManagerImpl.doGetUserInstance - This
   operation is not implemented !");
return null;
}

}
```

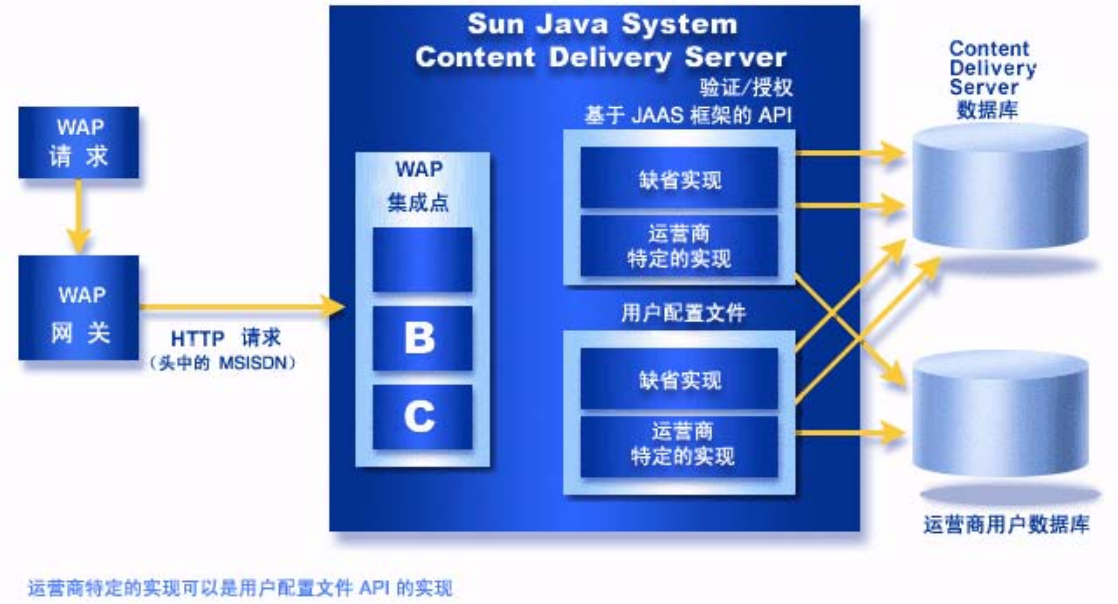
第 7 章

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特性。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 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、但与全面理解体系结构相关的组件。

图 7 WAP 网关适配器体系结构



WAP 网关适配器从特定 WAP 网关解析 HTTP 头并获得 MSISDN。

有关本节中未介绍的类或方法的信息，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。

有关本节中未介绍的方法信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 中的 WAP 网关 API 的 Javadoc 工具的 HTML 输出。

7.1.1 doHandle()

此方法用于确定是否已实现了此类的其他方法。如果 `method` 已实现，则此方法应返回 `true`。如果 `method` 未实现，将返回 `false`。

```
public abstract boolean doHandle(String method)
    throws WAPGatewayException
```

7.1.2 getMSISDN()

解析 HTTP 头后以字符串形式返回 MSISDN。

```
public abstract String getMSISDN(HttpServletRequest request)
    throws WAPGatewayException
```

7.1.3 getUniqueId()

解析 HTTP 头后以字符串形式返回唯一设备 ID。

```
public abstract String getUniqueId(HttpServletRequest request)
    throws WAPGatewayException
```

7.2 使用 WAP 网关 API

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

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

WAP 网关必须配置为将 MSISDN 或唯一设备 ID 转发到 Content Delivery Server。`WAPGatewayManager` 和 `WAPGatewayAdapter` 类可以从 `cdsapi.jar`（位于 `$CDS_HOME/deployment/`*deployment-name*`/lib/cdslib` 目录）中找到。要为任何其他 WAP 网关注册一个新类以扩展 `WAPGatewayAdapter` 类，请将类文件名添加到 `$CDS_HOME/deployment/`*deployment-name*`/conf` 目录中的 `wapgateway.config` 文件中。下面的语句是使用 Content Delivery Server 为 Nokia Activ Server 2.0.1 注册一个适配器的实例。

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

将适配器置于 `$CDS_HOME/deployment/`*deployment-name*`/lib/external` 目录以便 Content Delivery Server 能够在执行期间找到该文件。

7.3 示例

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

代码示例 10 WAPGatewayAdapter 类的样例扩展

```
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>");
  }

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

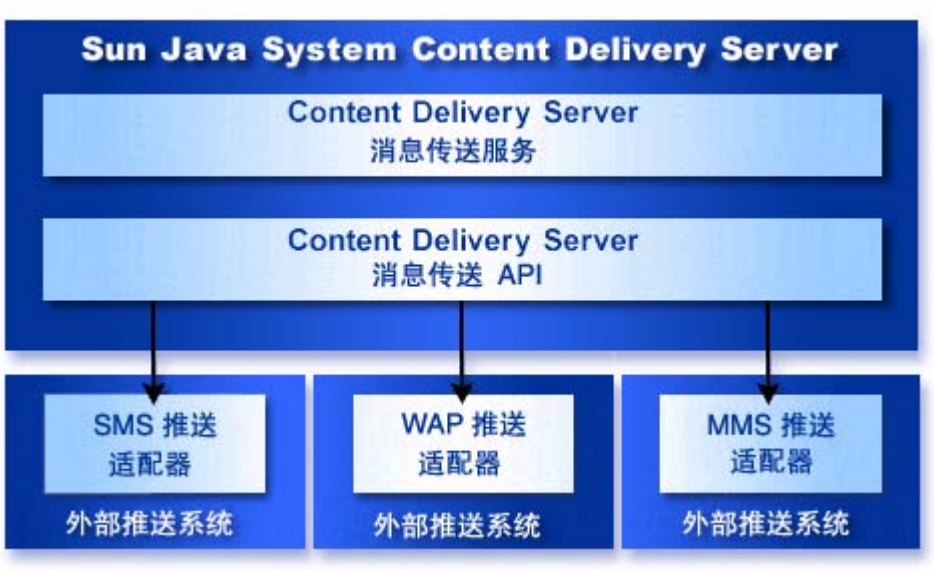
第 8 章

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集成自身的 WAP、 SMS 和 MMS 推送实现 （通过提供适配器进行集成）提供了一种机制。 Content Delivery Server 还提供了可以在大多数情况下使用的缺省 WAP、 SMS 和 MMS 推送实现。

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 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。

有关本节中未介绍的类或方法的信息，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。

```
public PushResponse pushMessage（PushMessage msg, int retryNum）
    throws PushMessageFailException;
```

8.2 PushMsgListener 接口

PushMsgListener 接口声明推送消息监听器实现所需的方法。

8.2.1 connect()

此方法在 SMSC 发送推送消息时由 Content Delivery Server 消息传送服务调用。

```
public boolean connect()
```

8.2.2 initialize()

这是 Content Delivery Server 消息传送服务调用的第一个方法，以便在连接到 SMSC 之前初始化 SMSC 参数。

```
public void initialize(String pushActionType) throws
    InitializationFailedException
```

8.2.3 listen()

当 PushMsgListener 已连接到 SMSC 并且希望监听来自设备的传入消息时，此方法将由 Content Delivery Server 消息传送服务调用。

```
public void listen()
```

8.2.4 sendKeepAliveMsg()

如果一段时间后 SMSC 要求发送 “保持活动”的信号，那么此消息将由 Content Delivery Server 消息传送服务调用。

```
public void sendKeepAliveMsg()
```

8.3 PushMessage 接口

PushMessage 是由 Content Delivery Server 生成的所有不同类型推送消息的接口。MMSPushMessage、WAPPushMessage、SMTPMessage 和 SMSMessage 类扩展了 PushMessage。

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。

8.3.1 addUserAgent()

设置接收订户设备的用户代理。

```
public void addUserAgent(String ua)
```

8.3.2 getAllUserAgents()

返回接收订户设备的所有用户代理。

```
public ArrayList getAllUserAgents()
```

8.3.3 getAttribute()

返回所有消息特性。

```
protected Object getAttribute(String attributeName)
```

8.3.4 getDestinationAddress()

返回目标地址，即接收订户的电话号码或者电子邮件 ID。

```
public String getDestinationAddress()
```

8.3.5 getJMSMessageId()

返回 JMS 消息的 ID。

```
public String getJMSMessageId()
```

8.3.6 getMessageText()

返回消息文本。

```
public String getMessageText()
```

8.3.7 getPushCategory

返回推送消息的类别。

```
public long getPushCategory()
```

8.3.8 getPushDomain()

返回推送消息的域。

```
public long getPushDomain()
```

8.3.9 getPushType()

返回消息类型。

```
public String getPushType()
```

8.3.10 getSubscriberId()

返回订户 ID。

```
public long getSubscriberId()
```

8.3.11 getUniqueDeviceId()

返回唯一设备 ID。

```
public String getUniqueDeviceId()
```

8.3.12 getVendingContentId()

返回销售内容 ID。

```
public long getVendingContentId()
```

8.3.13 setAllUserAgents()

设置接收订户设备的多个用户代理。

```
public void setAllUserAgents(ArrayList list)
```

8.3.14 setAttribute()

设置所有消息的特性。

```
protected void setAttribute(String attributeName, Object
    attributeVal)
```

8.3.15 setDestinationAddress()

设置推送消息的接收订户的目标地址。目标地址可以是电话号码或电子邮件 ID。

```
public void setDestinationAddress(String receiverId)
```

8.3.16 setJMSMessageId()

设置 JMS 消息的 ID。

```
public void setJMSMessageId(String msgId)
```

8.3.17 setMessageText()

设置消息文本。

```
public void setMessageText(String text)
```

8.3.18 setPushCategory()

设置推送消息的类别。

```
public void setPushCategory(long category)
```

8.3.19 setPushDomain()

设置推送消息的域。

```
public void setPushDomain(long domain)
```

8.3.20 setSubscriberId()

设置订户的 ID。

```
public void setSubscriberId(long subId)
```

8.3.21 setUniqueDeviceId()

设置设备唯一的 ID。

```
public void setUniqueDeviceId(String uniqueId)
```

8.3.22 setVendingContentId()

设置销售内容 ID。

```
public void setVendingContentId(long contentId)
```

8.3.23 toString()

将 `PushMessage` 对象的数据显示到日志。此方法主要用于调试。

```
public String toString()
```

8.4 SMSURLEncoder 接口

SMSURLEncoder 接口声明了获取 HTTP 请求的 URL 所需的方法。

8.4.1 encode()

此方法由 `HTTPSMSPushMessageSender` 调用，以获取服务提供商宿主特定 SIM 卡时 HTTP 请求所需的 URL 和参数。

```
public SMSHTTPMessageData encode(SMSMessage sm)
```

8.5 SMSMessage 类

`SMSMessage` 类扩展了 PushMessage，并代表了一个 SMS 推送消息。SMSMessage 包括的方法可以获取并设置消息内容的名称和类型以及 MIME 类型。还包括获取并设置用于下载的 URL 的方法。

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。

8.5.1 setContentType()

设置消息的内容类型。

```
public void setContentType( String contentType)
```

8.5.2 getContentType()

获取消息的内容类型。

```
public String getContentType()
```

8.5.3 setContentName()

设置内容或应用程序的显示名称。

```
public void setContentName(String cName)
```

8.5.4 getContentName()

获取内容或应用程序的显示名称。

```
public String getContentName()
```

8.5.5 getMimeType()

获取 MIME 类型。

```
public String getMimeType()
```

8.5.6 setMimeType()

设置 MIME 类型。

```
public void setMimeType(String mimeType)
```

8.5.7 setDownloadURL()

设置应用程序的下载 URL。此方法在 Content Delivery Server 需要向设备发送下载消息时由它调用。

```
public void setDownloadURL(String downloadUrl)
```

8.5.8 getDownloadURL()

如果在消息中设置了 URL，则获取下载 URL。如果未指定任何下载 URL 则返回空。

```
public String getDownloadURL()
```

8.6 WapPushMessage 类

`WapPushMessage` 类扩展了 WAP 支持的 `PushMessage` 接口。

8.6.1 setDownloadURL()

设置应用程序的下载 URL。此方法在 Content Delivery Server 需要向设备发送下载消息时由它调用。

```
public void setDownloadURL(String downloadUrl)
```

8.6.2 getDownloadURL()

获取在消息中设置的下载 URL。如果未指定下载 URL 则返回空。

```
public String getDownloadURL()
```

8.7 SMTPMessage 类

此类表示 SMTP 推送消息。 `SMTPMessage` 扩展了 `PushMessage`。 `SMTPMessage` 包括的方法可以获取并设置消息主题以及用来覆盖消息的 “发件人” 字段的值。

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。

8.7.1 getOverrideFrom()

返回用作电子邮件消息的 “发件人” 地址的地址。

```
public String getOverrideFrom()
```

8.7.2 getSubject()

获取 SMTP 消息的主题。

```
public String getSubject()
```

8.7.3 setOverrideFrom()

设置电子邮件消息的覆盖 “发件人” 地址。如果设置为非空的值，它将用于已发送电子邮件消息的标题。

```
public void setOverrideFrom(String value)
```

8.7.4 setSubject()

设置 SMTP 消息的主题。

```
public void setSubject(String subject)
```

8.8 ContentSlide 类

此类用于存储 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。

8.8.1 getContentData()

返回与 MMS 推送消息关联的二进制数据。

```
public byte[] getContentData()
```

8.8.2 getContentId()

返回与 MMS 推送消息的二进制数据关联的唯一 ID。

```
public String getContentId()
```

8.8.3 getContentMimeType()

返回 MMS 推送消息的二进制数据的 MIME 类型。

```
public String getContentMimeType()
```

8.8.4 setContentData()

设置 MMS 推送消息的二进制数据。

```
public void setContentData(byte[] contentData)
```

8.8.5 setContentId()

设置与 MMS 推送消息的二进制数据关联的唯一 ID。

```
public void setContentId(String contentId)
```

8.8.6 setContentMimeType()

设置 MMS 推送消息的二进制数据的 MIME 类型。

```
public void setContentMimeType(String contentMimeType)
```

8.9 MMSSlide 类

此类是 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 方法由服务器使用。所有 get 方法可以由推送消息的实现使用。

8.9.1 getAudioContent()

返回 MMSSlide 对象的音频内容。

```
public ContentSlide getAudioContent()
```

8.9.2 getImageContent()

返回 MMSSlide 对象的图像内容。

```
public ContentSlide getImageContent()
```

8.9.3 getTextContent()

返回 MMSSlide 对象的文本内容。

```
public ContentSlide getTextContent()
```

8.9.4 getVideoContent()

返回 MMSSlide 对象的视频内容。

```
public ContentSlide getVideoContent()
```

8.9.5 setAudioContent()

设置 MMSSlide 对象的音频内容。

```
public void setAudioContent(String contentId, String
    contentMimeType, byte[] contentData)
```

8.9.6 setImageContent()

设置 MMSSlide 对象的图像内容。

```
public void setImageContent(String contentId, String
    contentMimeType, byte[] contentData)
```

8.9.7 setTextContent()

设置 MMSSlide 对象的文本内容。

```
public void setTextContent(String contentId, String contentMimeType,
    byte[] contentData)
```

8.9.8 setVideoContent()

设置 MMSSlide 对象的视频内容。

```
public void setVideoContent(String contentId, String
    contentMimeType, byte[] contentData)
```

8.10 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。此类将包含 “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 (SMIL) 数据 （如果可用）。同时还封装了 MMSSlide 对象。

8.10.1 addMMSSlide()

设置 MMSSlide 对象。

```
public void addMMSSlide(MMSSlide mmsSlide)
```

8.10.2 addRecipient()

设置接收订户的电话号码或电子邮件 ID。

```
public void addRecipient(String to)
```

8.10.3 getAllMMSSlides()

返回与此 MMSPushMessage 关联的所有 MMSSlide 对象。

```
public ArrayList getAllMMSSlides()
```

8.10.4 getAllRecipients()

返回消息的所有收件人的电话号码或电子邮件 ID。

```
public ArrayList getAllRecipients()
```

8.10.5 getDeliveryReportRequired()

返回 DeliveryReportRequired 特性的值。

```
public boolean getDeliveryReportRequired()
```

8.10.6 getFromAddress()

返回发送者的电话号码或电子邮件 ID。

```
public String getFromAddress()
```

8.10.7 getMessageClass()

返回 MessageClass 特性的值。

```
public String getMessageClass()
```

8.10.8 getMessagePriority()

返回 MessagePriority 特性的值。

```
public String getMessagePriority()
```

8.10.9 getReadReportRequired()

返回 ReadReportRequired 特性的值。

```
public boolean getReadReportRequired()
```

8.10.10 getSenderVisibility()

返回 SenderVisibility 特性的值。

```
public String getSenderVisibility()
```

8.10.11 getSMILPresentation()

返回 SMIL 数据。

```
public byte[] getSMILPresentation()
```

8.10.12 setDeliveryReportRequired()

设置 DeliveryReportRequired 特性的值。

```
public void setDeliveryReportRequired(boolean
   deliveryReportRequired)
```

8.10.13 setFromAddress()

设置发送者的电话号码或电子邮件 ID。

```
public void setFromAddress(String from)
```

8.10.14 setMessageClass()

设置 MessageClass 特性的值。

```
public void setMessageClass(String messageClass)
```

8.10.15 setMessagePriority()

设置 MessagePriority 特性的值。

```
public void setMessagePriority(String messagePriority)
```

8.10.16 setReadReportRequired()

设置 ReadReportRequired 特性的值。

```
public void setReadReportRequired(boolean readReportRequired)
```

8.10.17 setSenderVisibility()

设置 SenderVisibility 特性的值。

```
public void setSenderVisibility(String senderVisibility)
```

8.10.18 setSMILPresentation()

设置 SMIL 数据。

```
public void setSMILPresentation(byte[] smil)
```

8.11 PushResponse 类

PushResponse 是由外部推送服务生成的所有不同类型推送响应的基类。所有 set 方法由推送消息的发送器实现使用。所有 get 方法可以由服务器使用以便将推送消息记录到数据库中。

8.11.1 getMessageId()

返回消息 ID。

```
public String getMessageId()
```

8.11.2 getResponseDescription()

返回响应的说明。

```
public String getResponseDescription()
```

8.11.3 getResponseStatus()

返回响应的状态。

```
public String getResponseStatus()
```

8.12 PushConstants 类

PushConstants 类包含所有推送服务服务器支持的常量。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。

PushType 常量

```
public static final long SMS_PUSH_TYPE = "sms"
public static final long WAP_PUSH_TYPE = "wap"
public static final long MMS_PUSH_TYPE = "mms"
public static final long SMTP_PUSH_TYPE = "smtp"
```

Push 类别类型常量

```
public static final long PUSH_SUBSCRIBER_DETAIL_URL_CATEGORY =
    "subscriber_detail_url"
public static final long PUSH_MOBILE_ORIGINATED_DETAIL_URL_CATEGORY
    = "mobile_originated_detail_url"
public static final long PUSH_CONTENT_BINARY_CATEGORY =
    "content_binary"
public static final long PUSH_GIFTING_DETAIL_URL_CATEGORY =
    "gifting_detail_url"
public static final long PUSH_CONTENT_SHARING_DETAIL_URL_CATEGORY =
    "content_sharing_detail_url"
public static final long PUSH_PASSWORD_REMINDER_CATEGORY =
    "password_reminder"
public static final long PUSH_CAMPAIGN_MESSAGE_CATEGORY =
    "campaign_message"
public static final long PUSH_DEVELOPER_MESSAGE_CATEGORY =
    "developer_message"
public static final long PUSH_NEW_MODEL_EMAIL_NOTIFICATION_CATEGORY
    = "new_model_email_notification"
```

PushDomain 常量

```
public static final long SUBSCRIBER_PUSH_DOMAIN = "subscriber"
public static final long VENDING_PUSH_DOMAIN = "vending"
public static final long CATALOG_PUSH_DOMAIN = "catalog"
public static final long DEVELOPER_PUSH_DOMAIN = "developer"
```

8.13 使用消息传送 API

推送消息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。

代码示例 11 样例 pushsenderfactory.xml 文件

```
<?xml version="1.0" encoding='UTF-8' ?>
<PushSenderConfig nodeid="0">
  <pushmsgsenderset nodeid ="1">
    <pushmsgsender nodeid ="2" class =
"com.sun.content.server.server.msgserver.push.SMSHTTPPushMsgSender
   " protocol="sms"/>
    <pushmsgsender nodeid ="3" class =
"com.sun.content.server.server.msgserver.push.WAPPushMsgSender"
   protocol="wap"/>
    <pushmsgsender nodeid ="4" class =
"com.sun.content.server.server.msgserver.push.SMTPushMsgSender"
   protocol="smtp"/>
</pushmsgsenderset>
</PushSenderConfig>
```

此文件中注册了三个适配器。请指定适配器支持全限定类名和协议。例如，如果适配器用于 SMS 推送，则协议应为 SMS。确保在类路径中设置了适配器类和相关的类。

8.14 MMSSender 接口

MMSSender 接口声明用于发送 MMS 消息的方法。此接口必须在集成过程中由供应商特定的多媒体消息服务中心 (MMSC) 实现。

8.14.1 sendMMS()

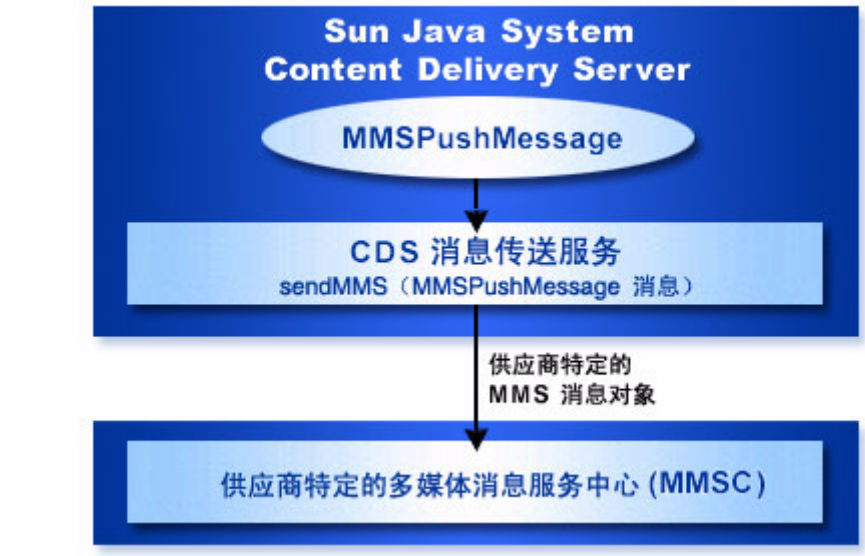
发送 MMS 消息的功能封装在 sendMMS 方法中：

```
sendMMS(com.sun.content.server.server.messaging.message.MMSPushMes
   sage message)
```

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` 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 9 发送 MMS 消息的进程流



实现 `MMSSender.sendMMS` 供应商特定版本的代码包含以下常规步骤：

1. **连接到供应商特定的 MMSC。**
2. **实现 `MMSSender.sendMMS` 执行下列任务：**
   a. **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。**
   b. **将新的消息对象发送到供应商特定的 MMSC。**

第 9 章

确认服务 API

通过 Sun Java System Content Delivery Server 确认服务 API， Content Delivery Server 可以处理多媒体消息服务中心 (MMSC) 发送的的确认信息。确认信息一般在内容下载到设备后发送。

确认服务 API 包括

- `ConfirmServiceAdapter` 类：一个抽象类，您可以扩展该类以连接到 MMSC 并监听消息。
- `ConfirmResponse` 类：包含收到的确认信息的类。
- `ConfirmServiceException` 类：确认服务 API 抛出的异常。

有关这些类的其他信息，请参见 `$CDS_HOME/javadoc/cdsapi/index.html` 处的 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多媒体消息服务 (MMS) 标准的设备。当设备收到 MMS 消息内容时，确认消息将通过 MMSC 发回。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。

9.2 ConfirmServiceAdapter 类

`ConfirmServiceAdapter` 类使用 MMSC 建立连接、监听确认消息以及将收到的消息传递到 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` 包中。

9.2.1 connect()

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

使用此方法可以将 Content Delivery Server 连接至正在使用的 MMSC。

9.2.2 listen()

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

使用此方法可以监听来自 MMSC 的确认消息。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 11 ConfirmResponse 参数

参数	说明
pushType	收到的消息的类型。该值必须是在 PushConstants 类中定义的类型之一（请参见第 100 页上的第 8.12 节 “PushConstants 类”）。
messageID	由 MMSC 指定以标识消息的 ID。
responseStatus	响应的状态。
responseDescription	响应的说明。
responseObject	不用于 MMS 消息。传递空值。

9.2.3 messageReceived()

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

使用此方法可以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

在 $CDS_HOME/deployment/*deployment-name*/conf 目录中修改 ConfirmListener.properties 文件来添加以下属性的另一个实例：

```
confirmservice.class.id
```

将此属性设置为 ConfirmServiceAdapter 的实现的类名。例如，

```
confirmservice.class.id=com.sun.content.server.server.confirm.mms.
    MMSConfirmService
```

确认服务 API 的类可以在 cdsapi.jar （位于 $CDS_HOME/deployment/*deployment-name*/lib/cdslib 目录）中找到。

编译适配器时， cdsapi.jar 文件必须位于类路径中。

为使适配器可用于 Content Delivery Server，请执行以下操作：

1. **为适配器创建 Java 归档 (JAR) 文件，并将 JAR 文件置于** $CDS_HOME/deployment/***deployment-name***/lib/external **目录中，以便在执行过程中 Content Delivery Server 可以找到该文件。**

   Content Delivery Server 需要重新启动才能发觉新的 JAR 文件。

2. **在** $CDS_HOME/deployment/***deployment-name***/conf **目录中修改** ConfirmListener.properties **文件来添加** confirmservice.class.id **属性的另一个实例：**

   将此属性设置为 ConfirmServiceAdapter 的实现的类名。例如，

   ```
   confirmservice.class.id=com.sun.content.server.server.confirm.mms.
       MMSConfirmService
   ```

第 10 章

订户 API

通过 Sun Java System Content Delivery Server 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。

以 Java 编程语言编写，位于具备 Content Delivery Server Subscriber Portal 的服务器上的客户机可以直接调用订户 API。如果编写客户机的语言不是 Java，或者客户机所在的服务器不包含 Content Delivery Server Subscriber Portal，那么客户机可以通过 XML-RPC （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 （例如语言环境、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、各个项目的详细资料和订户标记的内容。
- `ApiUtil`：此类提供多种功能，例如启动事务或检查数据库版本。
- `IApiContext`：此接口为当前订户提供信息。
- `ICategoryService`：此接口提供对类别树和订户类别列表的访问。
- `IContentService`：此接口提供对内容的访问，以便完成浏览、查找、检索和购买。
- `IDownloadService`：此接口提供对内容描述符的访问。
- `IGiftingService`：通过此接口订户可以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。
- `ISystemService`：此接口提供对系统级内容的访问，例如语言环境、内容类型和设备型号。
- `IUserService`：此接口提供对有关订户信息的访问并且可以创建新的订户帐户。
- `CDSException`：此类是出现错误时订户 API 抛出的异常。

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

10.1 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 Java 类。如果您使用 XML-RPC 访问 Content Delivery Server 数据，则请参见第 112 页上的第 10.3 节 “XML-RPC 实现”以获得等效的处理程序信息。

通常，您为订户接口创建的客户机应用程序应包含多个操作，如下面列表中所述。

- 调用 `ApiContextFactory` 创建 `IApiContext` 对象。

  `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，并将该对象存储在 `HttpSession` 对象中。有关样例代码，请参见第 110 页上的第 10.2.2 节 “IApiContext 对象创建示例”。

- 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。要创建服务，必须具有 `IApiContext` 对象。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 111 页上的第 10.2.3 节 “创建服务的示例”。

- 调用所创建的服务方法检索要使用的信息。

  例如，如果创建了 `IContentService`，则可以调用 `getCampaigns()` 获得订户可用活动列表，或调用 `getPurchases()` 获得订户已购买项目列表。

10.2 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 `subscriberportal.jar` 文件中，该文件位于以下某个位置：

- 对于 Sun Java System Application Server，该文件位于 `$CDS_HOME/deployment/`*deployment-name*`/sun/domains/`*server-domain*`/`*server-name*`/applications/j2ee-modules/CDSSubscriberPortal_1/WEB-INF/lib`。
- 对于 WebLogic Server，该文件位于 `$CDS_HOME/deployment/`*deployment-name*`/weblogic/domains/`*server-domain*`/applications/subscriberportal/WEB-INF/lib`。

其中，*deployment-name* 是部署 Catalog Manager 时指定的名称，*server-domain* 是在 `app.server.domain` 属性的配置文件中指定的值，*server-name* 是在 `app.server.name` 属性的配置文件中指定的值。

如果您的客户机应用程序是 Java 应用程序，那么请使用 `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。

要成功执行，包含客户机的 JAR 文件必须置于 `$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 Content Delivery Server Subscriber Portal 不在同一台服务器上，请参见第 112 页上的第 10.3 节“XML-RPC 实现”获取有关通过 XML-RPC 访问订户 API 的信息。

10.2.1 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()` 指明事务的起点。**
2. **如果事务成功完成，请调用 `ApiUtil.commitTransaction()` 提交完成的工作。如果处理事务的过程中出错，请调用 `ApiUtil.rollbackTransaction()` 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()` 的 `finally` 块中释放。**

有关样例实现，请参见第 110 页上的第 10.2.2 节“IApiContext 对象创建示例”和第 111 页上的第 10.2.3 节 “创建服务的示例”中的代码示例。

10.2.2 IApiContext 对象创建示例

以下节选代码显示如何创建 IApiContext 对象。

代码示例 12 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 in the HttpSession
  session.setAttribute("API_CONTEXT", apiContext);

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

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

10.2.3 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。

代码示例 13 创建服务

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

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

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

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

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

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

10.3 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，则客户机需要使用 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。

10.3.1 设置到 Content Delivery Server 的访问

要从 Content Delivery Server 获取数据，客户机必须可以与 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` 属性包含主机的列表，指明了哪些主机的请求将被接受。

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
```

10.3.2 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，应用程序的常规流 （请参见第 108 页上的第 10.1 节 “一般处理流程”）都相同。 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：

- 调用 XML-RPC 方法的指导
- AuthenticationHandler
- CategoryHandler
- ContentHandler
- DownloadHandler
- GiftingHandler

- MessageHandler
- SystemHandler
- UserHandler
- 方法的参数

10.3.2.1 调用 XML-RPC 方法的指导

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 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。返回散列表。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 134 页上的第 10.3.3 节 “使用处理程序的示例”中提供示例。

10.3.2.2 AuthenticationHandler

`AuthenticationHandler` 等效于 `APIContextFactory` 类。该处理程序创建包含订户特征的对象。调用处理程序中的方法的指导在第 113 页上的第 10.3.2.1 节 “调用 XML-RPC 方法的指导”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` 方法。

表 12 AuthenticationHandler 的方法

方法名称	说明	参数	返回 [1]
getAnonymousApiContext	为匿名订户创建 APIContext 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 APIContext 对象。	下列项目之一： • username 和 password • mobileId • uniqueId • subscriberId	apiContext

[1] 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。

10.3.2.3 CategoryHandler

`CategoryHandler` 等效于 `ICategoryService` 接口。通过此处理程序可以访问类别树和订户类别列表。调用处理程序中的方法的指导在第 113 页上的第 10.3.2.1 节 “调用 XML-RPC 方法的指导”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` 方法。

表 13 CategoryHandler 的方法

方法名称	说明	参数	返回 [1]
getCategory	获取当前订户的指定类别。	apiContext、categoryId、includeContentCount	category
getNotEmtpySubCategories	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
hideCategory [2]	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 [2]	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 [2]	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 [2]	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

[1] 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。
[2] 该方法更改内存中维护的类别列表。要永久更改，必须调用 `updateCategories`。

10.3.2.4 ContentHandler

`ContentHandler` 等效于 `IContentService` 接口。通过此处理程序可以访问内容，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 113 页上的第 10.3.2.1 节 “调用 XML-RPC 方法的指导”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` 方法。

表 14 ContentHandler 的方法

方法名称	说明	参数	返回 [1]
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。	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
deleteBookmark	从订户的书签内容列表删除某项内容。	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	获取束中的内容项。	apiContext、contentId	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 14　ContentHandler 的方法 (*续*)

方法名称	说明	参数	返回 [1]
getContentDetails	获取某项内容的信息。	apiContext、contentId、campaignId （可选）、 bundleId （仅当内容是束的一部分才指定）、retailPrice、 filter	content
getContentDetailsList	获取 `contentIdList` 中指定的每项内容的信息。	apiContext、contentIdList、campaignId （可选）、 bundleId （仅当内容是束的一部分才指定）、retailPrice、 filter	contentList
getContentSummary	获取某项内容的概要信息。	apiContext、contentId	contentSummary
getPurchasedBundles	获取订户购买的包含 `contentId` 中指定内容项的束列表。	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
isBookmarked	确定订户是否已对内容编制书签。	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	确定是否内容可以使用 MMS 推送到设备。	apiContext、contentId	isMMSCapable
purchaseContent	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
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。）	apiContext、categoryId、contentTypeIdList （可选）、keyword、startIndex、numberToReturn	contentList、 totalSize

[1] 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。

10.3.2.5 DownloadHandler

`DownloadHandler` 等效于 `IDownloadService` 接口。通过此处理程序可以访问下载内容所需描述符。调用处理程序中的方法的指导在第 113 页上的第 10.3.2.1 节 “调用 XML-RPC 方法的指导”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` 方法。

表 15 DownloadHandler 的方法

方法名称	说明	参数	返回 [1]
downloadConfirm	确认是否内容已下载到设备。	apiContext、codedTicket、isOneStepConfirm、status	无
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	删除某设备的内容。	apiContext、codedTicket、isOneStepConfirm、status	无
downloadJAD	创建 Java 应用程序描述符 (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	使用多媒体消息服务 (MMS) 将内容推送到设备。	apiContext、categoryIds、categoryList	categoryList

[1] 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。

10.3.2.6 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 113 页上的第 10.3.2.1 节 “调用 XML-RPC 方法的指导”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 方法。

表 16 GiftingHandler 的方法

方法名称	说明	参数	返回 [1]
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 所指定礼品的有关信息。	apiContext、 giftId、 filter、 bundledContentId	gifting
getGiftingByTicket	获取 giftTicket 所指定礼品的有关信息。	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	确定是否收件人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

[1] 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。

10.3.2.7 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 113 页上的第 10.3.2.1 节 “调用 XML-RPC 方法的指导”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 方法。

表 17 MessageHandler 的方法

方法名称	说明	参数	返回 [1]
sendMessageToMobileId	将消息发送到指定的号码	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

[1] 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。

10.3.2.8 SystemHandler

SystemHandler 等效于 ISystemService 接口。通过此处理程序可以访问系统级数据，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 113 页上的第 10.3.2.1 节 “调用 XML-RPC 方法的指导”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` 方法。

表 18 SystemHandler 的方法。

方法名称	说明	参数	返回 [1]
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
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
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。	apiContext、subscriberId、mobileId、 contentId（可选）、 eventType	无

[1] 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。

10.3.2.9 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 113 页上的第 10.3.2.1 节 “调用 XML-RPC 方法的指导”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` 方法。

表 19 UserHandler 的方法

方法名称	说明	参数	返回 [1]
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
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

表 19 UserHandler 的方法 *(续)*

方法名称	说明	参数	返回 [1]
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

[1] 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。

10.3.2.10 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 20 方法参数

参数	类型	说明
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 • uniqueId • modelId • isAnonymous • isProvisioned • isRegistered • passwordRequiresReset • isCategoryCustomized • roleId • propertyMap **注意**：对于匿名订户，仅包含 subscriberId、 `localeCode` 和 `modelId`。
bundledContentId	整型	束中由 Content Delivery Server 指定给单个项的内部 ID。
bundleId	整型	Content Delivery Server 为当前使用的束指定的内部 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。
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。
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。

表 20 *方法参数（续）*

参数	类型	说明
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
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。
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 • name • shortDescription • longDescription • submitDate • sizeInKB • version • contentType • numberOfDownloads • smallIconUrl • largeIconUrl • userGuideUrl • previewUrl • screenShot1Url • screenShot2Url • developerName • developerUrl • downloadTimes • networks • downloadUrl • isBookmarked • isPurchaseRequiredBeforeDownload • isUnsubscribeAvailable • isValidOnCurrentModel • pricingDetails
contentBinary	字符串	内容的二进制格式。
contentCount	整型	类别中内容的项目数。

表 20 方法参数（*续*）

参数	类型	说明
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。
contentLength	整型	内容的大小。
contentList	向量，contentSummary 类型的元素	内容项的列表。
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 • shortDescription • submitDate
contentType	字符串	使用内容的类型。该类型必须是 contentTypeList 中定义的某一项内容类型。
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。
description	字符串	对象的说明。根据所调用的方法，该参数可以是类别、活动、设备或语言环境的说明。
descriptorData	字符串	内容描述符二进制代码。
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。 0 表示列表顶端。
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 向量中的元素，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。

表 20 *方法参数 (续)*

参数	类型	说明
emailAddress	字符串	订户的电子邮件地址。
eventType	字符串	使用事件的类型。有效值是： • sms_request_for_campaign：表示事件是有关活动信息的请求。 • sms_request_for_content：表示事件是有关内容项信息的请求。
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` 时的有效标志： • filterDetailsDownload：表示是否返回下载信息。 • filterDetailsDownloadCount：表示是否返回下载计数。 • filterDetailsIsBookmarked：表示是否返回书签上的信息。 • filterDetailsResourceURLs：表示是否返回资源 URL。 • filterDetailsPricingAndPurchase：表示是否返回价格和购买信息。 • filterDetailsIsValidOnCurrentModel：表示是否返回内容可否在设备上执行方面的信息。 使用 `GiftHandler` 时的有效标志： • filterGiftsContent：表示是否返回有关内容的信息。 • filterGiftsDownload：表示是否返回下载信息。
firstName	字符串	订户的名字。
gender	字符串	订户的性别。有效值是： • M：表示订户是男性。 • F：表示订户是女性。
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。
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。

表 20 *方法参数（续）*

参数	类型	说明
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。有效值是： • Gift Purchased：表示礼品已付款。 • Gift Download Initiated：表示接收订户已开始下载礼品。 • Gift Download Confirm：表示礼品已成功下载。 • Gift Expired：表示礼品已过期。 • Gift Used：表示接收订户已使用礼品。
giftTicket	字符串	内部对象，用于验证订户是否可访问礼品。
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	表示是否应该计算类别中项目数的标志。 **True** 表示应计算该数目。 **False** 表示不应计算该数目。
isActive	布尔型	表示是否将该类别显示给订户的标志。 **True** 表示显示类别。 **False** 表示不显示类别。
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。
isBookmarked	布尔型	表示订户是否已对内容编制书签的标志。 **True** 表示已对内容编制书签。 **False** 表示未对内容编制书签。

表 20 *方法参数 (续)*

参数	类型	说明
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。
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	表示是否内容是免费的标志。 **True** 表示内容是免费的。 **False** 表示必须购买内容。
isGiftExpired	布尔型	表示是否礼品已过期的标志。 **True** 表示礼品已过期，无法再领取。 **False** 表示礼品未过期。
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。
isOneStepConfirm	布尔型	表示下载是单步还是两步。 **True** 表示单步下载。 **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一个条目。 **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	表示是否设备是可推送的标志。 **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** 表示跳过试用，订户可以立即购买内容。 **False** 表示不应跳过试用。
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。

表 20 *方法参数（续）*

参数	类型	说明
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。
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。
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说明。
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。
message	字符串	要发送给订户的消息。
messageCategory	整型	要发送消息的类别。类别 1 到 7 发送到订户，类别 9 发送到 Catalog Manager 管理员。有效值是： • 1: 表示消息包含指向有关内容项详细资料的 URL。 • 2: 表示消息源于移动消息，包含指向有关内容项详细资料的 URL。 • 3: 表示消息包含二进制内容。 • 4: 表示消息包含礼品，以及指向有关内容项详细资料的 URL。 • 5: 表示消息包含通知，以及指向有关内容项详细资料的 URL。 • 6: 表示消息包含密码提醒。 • 7: 表示消息包含有关活动的信息。 • 9: 表示已有新设备添加到 Content Delivery Server。
messageType	整型	要发送消息的类型。有效值是： • 1: 表示消息被发送到订户设备。 • 2: 表示消息被发送到订户的电子邮件。
middleName	字符串	订户的中间名。
mimeType	字符串	内容的 MIME 类型。

表 20 *方法参数 (续)*

参数	类型	说明
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。
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。返回所有项目，请指定为 -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`** 表示密码 无需被重置。
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。该容器包含以下各项： • notifyPromotions 要删除某个首选项，将相应的值设置为空字符串 ('')。
previewUrl	字符串	指向内容预览文件的 URL。

表 20 方法参数（*续*）

参数	类型	说明
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 • isDownloadRecurring • subscriptionPrice • subscriptionFrequency • isSubscriptionRecurring • trialCount • usagePrice • usageCount • isFree • isTrialAvailable
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。这些参数是 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
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
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。

表 20 *方法参数 (续)*

参数	类型	说明
retailPrice	布尔型	表示返回哪一种价格的标志。 **True** 表示返回当前零售价格。 **False** 表示返回订户支付价格。如果订户购买后零售价格发生变化，那么这些价格就不同。
roleId	整型	指定给订户的角色。有效值是： • 0: 测试订户。表示订户仅可以访问状态为 Testing 的内容。 • 1: 订户。表示订户具有标准特权。
salutation	字符串	尊称，例如先生。
screenShot1Url	字符串	指向内容的第一幅屏幕快照的 URL。
screenShot2Url	字符串	指向内容的第二幅屏幕快照的 URL。
shortDescription	字符串	来自内容有关信息的简短说明。
sizeInKB	字符串	内容的大小。
smallIconUrl	字符串	指向内容小图标的 URL。
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 / 市 / 自治区。
status	字符串	确认设备返回的状态字符串。
statusList	向量，整型类型的元素	内容状态的列表。以下为有效状态： • 1: 活动。表示已存储内容，并且订户可以使用。 • 2: 不活动。表示已存储内容，但是订户无法使用。 • 3: 不可用。表示订户无法使用内容。 • 4: 测试。表示正在测试内容，只有指定了测试者角色的订户才可以使用。
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**、 **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。

表 20 *方法参数 (续)*

参数	类型	说明
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。
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。

10.3.3 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。

10.3.3.1 ApiContext 对象创建示例

以下节选代码显示如何创建 `ApiContext` 对象。该样例使用 Java 编程语言的绑定。

代码示例 14 创建 APIContext 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");
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);
  ...
}

...
```

10.3.3.2 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。

代码示例 15 创建处理程序

```
...
// Get a reference to the XmlRpcClient
String url = "http://[[cds server host]]:[[cds server port]]/
   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);
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
  ...
}
...
```

索引

E

F

G

H

I

J

K

L

M

N

P

Q

R

S

T

U

V

W

X

Y

Z