2018

THE IOS MDM PROTOCOL

iOS 移动设备管理协议

《BH_US_11_Schuetz_InsideAppleMDM_WP.pdf》翻译及改编版

原作者：David Schuetz

移动互联百科 http://www.baike.net

by 江哥一直在

目录

iOS 移动设备管理协议 1

摘要 4

介绍 4

申明 5

MDM 协议概述 5

基本 5

注册 5

创建简单的 MDM 服务器 7

要求 7

设置 APNS 8

MDM 注册配置文件 9

注册交换 10

发送推送通知 13

设备响应推送 13

设备锁定 14

清除密码 16

结论 17

附录 A - 命令列表 18

命令格式 18

设备响应 19

错误消息 21

设备锁定 22
擦除设备 22
清除密码 22
安全信息 22
安装的应用程序列表 23
设备信息 23
证书列表 24
描述文件列表 25
预植描述文件列表 26
限制列表 26
安装配置文件 26
删除配置文件 27
安装预植描述文件 27
删除预植描述文件 27
设备认证 27
令牌更新 28
附录 B - 源代码 29
MDMServer 截图 29
代码地址 31
技术咨询及服务 31
附录 C - 参考文献 31

摘要

移动设备管理（MDM）已经成为一个热门话题，并融入到了企业社会的各个层面，远程管理和控制移动设备的需求日益迫切。 Apple 公司通过为设备安装配置（描述）文件方式，为企业提供设备管理功能，并从 2010 年开始提供完整的 MDM 支持。不幸的是，通过 MDM 提供的确切功能以及协议本身的细节尚未由 Apple 公开发布。本文是移动互联技术博客作者（江哥一直在）翻译的由 Intrepidus 公司的 David Schuetz 作者整理的关于 iOS MDM 协议的 PDF 文档《BH_US_11_Schuetz_InsideAppleMDM_WP.pdf》，翻译的不足之处，敬请指正，如有任何版权问题，请邮件至 459104018@qq.com。

本文描述了 Apple 的 MDM 系统如何工作。详细介绍了 MDM 服务器与受管设备之间连接的方法，设备如何通过 MDM 服务器进行注册以及各种命令之间的交互。本文的每个命令都提供了完整的参数，以及设备的响应的详细信息。最后，移动互联百科网为广大的 MDM 开发者提供了一套非常简单的、开源的 MDM 服务器端源代码(OpenMDMServer),在 Github 上的连接为：https://github.com/keaijohnee/OpenMDMServer，作者也希望将自己的学习成果开源出来帮助中国开发者快速理解和搭建 MDM Server 开发和生产环境。

介绍

使用 iOS 设备（如 iPhone，iPad 或 iPod）会给企业带来严重的风险。这些移动设备是功能强大的计算机，具有很高的存储容量，企业数据可能存在潜在地超出控制范围，并有意或无意地将其泄露给未经授权的第三方。 iOS 上的许多保护应用专注于设备本身，例如使用密码来防止第三方应用或者设备获取本设备上的数据。然而，管理这种无形资产和数据对大型企业来说是一个严峻的挑战，而且也一直是一个复杂而繁琐的过程。

在 2010 年，Apple 推出了针对 iOS 的移动设备管理（Mobile Device Management，MDM）

服务，这是面向企业的面向 iOS MDM 问题的解决方案。该系统具有远程安装配置文件，查询设备设置以及某些远程控制的功能，如：设备锁定，解锁和远程擦除等。

申明

本文是在不使用任何 Apple 保密材料的情况下进行的，并受到非披露协议的约束。

MDM 协议概述

MDM 的一个关键特性是它允许管理员无需任何手动干预即可将配置文件推送到设备。

基本

MDM 服务主要由三个要素组成：

1、设备管理（iPhone，iPad，iPod）

2、服务器管理（各种 MDM 服务器）

3、服务器启动设备（APNS）

注册

MDM 服务器支持无线（OTA）注册， 在注册期间，设备向服务器提供唯一标识（UDID）信息，服务器将使用该标识信息通过 Apple 推送通知服务发送消息。从服务器到客户端或客户端到服务器的长期连接在 MDM 设计中是不存在的，目前仅与 APNS 的连接。这个长期的 APNS 连接是推送通知框架的一部分，支持多个 iOS 应用程序，而不仅仅是 MDM。

注册后，客户端设备和 MDM 服务器之间的每次交互都由四个元素组成：

1、服务器 MDMServer 请求苹果 APNS 推送服务

2、Apple 推送消息到设备

3、设备连接 MDMServer 服务器

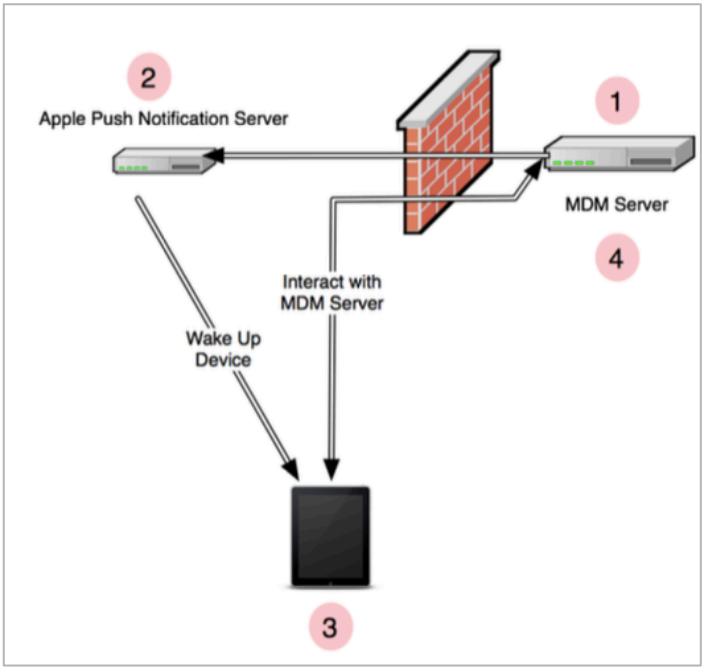
4、服务器发出命令和响应

推送通知

当 MDM 服务器需要与设备进行通信时，它将所需的命令排队，然后通过 APNS 发送一个非常简单的推送通知消息。此消息中不存在除识别标记之外的其他信息。收到通知后，设备联系服务器 ,然后将排队的命令提供给客户端。在完成该命令后 ,客户端设备将结果返回给 MDMServer，并关闭客户端和服务器之间的连接。

客户端/服务器交互

设备通过连接到指定的 URL 并以 Apple Property List（.plist）文件的形式交换 XML 格式的数据来与服务器进行交互。客户端连接到服务器时发送的第一条消息是一个简单的“状态：空闲”通知，指示设备已准备好接收来自 MDM 服务器的命令。收到此消息后，服务器发送可能正在等待的任何命令该设备。这个命令也是以 XML 格式的.plist 文件形式呈现，在大多数情况下只是一个简单的命令(可能有一些相关的参数)。设备根据命令行事 ,然后可以用另一个.plist 响应，提供对命令的简单确认，错误消息或详细响应（在设备清单和类似命令的情况下)。



创建简单的 MDM 服务器

本文提供一个非常基本的 MDM 服务器，代码是基于 Java 语言，框架使用：SpringMVC + Hibernate + MySQL。 本节将详细描述这样的服务器，完整的源代码将在《附录 B》中提供。

要求

【APNS】: 最重要的要求是获得 APNS 推送证书。 过去，这种证书只适用于 Apple Enterprise Developer 计划的成员（年费为 299 美元）。

【Java】: 此演示服务器基于 SpringMVC + Hibernate + MySQL，使用 JDK1.7 版本搭建。 服务器使用 OpenSSL 开发命令。

【网络连接】: 要发送推送通知，服务器需要与 Apple 的 APNS 服务器进行通信：这需要通过端口 2195 与 gateway.push.apple.com 进行出站 TCP 连接。同样需要管理的设备需要通过出站 TCP 端口 5223 连接到 APNS 最后，设备需要能够通过 MDM 注册配置文件中定义的任何端口连接 MDM 服务器本身。

设置 APNS

首先，必须获得 Apple 推送证书。 这是允许 MDM 服务器与客户端进行通信的方式，如果没有它(以及 Apple 提供的相应证书),MDM 服务将无法工作。在我的例子中 ,它被命名为“APSP：<uuid>”，其中 uuid 看起来像一串长长的十六进制数字（实际上就是这样）。 突出显示证书，并将其导出到.p12 文件（例如，PushCert.p12）。 程序会提示为密码保护私钥。

接下来，打开终端窗口，导航到保存证书的文件夹，并将文件转换为.pem 格式：例如，使用 openssl pkcs12 -in PushCert.p12 -out PushCert.pem。 从钥匙串导出密钥时使用的密码将需要打开.p12 文件，并且需要一个新的密码来保护.pem 文件。由于这个.pem 是加密的，所以每次尝试发送推送通知时，该工具都会提示输入密码。 为避免这种情况，请首先创建一个没有密码的密钥副本 :openssl rsa -in PushCert.pem -out PlainKey.pem。然后在文本编辑器中，将 PushCert.pem 中加密的 RSA 私钥部分替换为 PlainKey.pem 的内容。 完成此操作后，脚本将能够访问证书和密钥，以便在不使用密码的情况下发送通知。 当然，要确保这个文件很好的保护不被泄露，否则其他人将能够伪造你的推送通知。如下图：

MDM 注册配置文件

下一步是在 iPhone Configuration Utility2（IPCU）中创建一个配置文件，该配置文件将指示设备连接到 MDM 服务器。 在 IPCU 内部，创建一个新的配置文件，并选择 MDM 有效负载。在“服务器”字段中，输入服务器的 URL（例如 http://192.168.1.1/server），这将是设备用于轮询服务器以获取 MDM 命令的主要 URL，或者可以输入不同的 URL 在“检入”字段中（用

于初始连接到 MDM 服务器)示例服务器应该能够支持通过服务器 URL 进行检入 ,但在测试中，这被分隔到 http://192.168.1.1/checkin。

主题字段需要包含 APNS 推送证书的“使用者名称”部分中列出的用户 ID。 最后，需要生成设备的身份证书，并通过 IPCU 证书有效负载进行安装，并在此处进行选择。 这可以是使用钥匙串访问“证书助理”生成的简单证书，但请记住，它可能需要由可信实体签名。 如果没有，则用于签署身份的证书颁发机构（CA）可能需要安装在设备上，以便设备识别它。 （尽管如前所述，测试服务器甚至没有使用该证书)。

完成所有工作后，保存配置，然后可以通过 USB（或无线方法）将其复制到用于服务器测试的 iOS 设备。

注册交换

在安装 MDM 配置文件时，设备将尝试连接到 MDM 服务器。 它首先向 Check In URL 发送一个基本的“Authenticate”请求，提供设备的请求通用设备标识符（UDID）和推送通知主题。此交换为服务器提供了一个接受或拒绝注册请求的机会，该注册请求基于 Topic 和（更可能）UDID。

```
PUT: /checkin

    <?xml version="1.0" encoding="UTF-8"?>

    <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"

    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">

    <plist version="1.0">

    <dict>

        <key>MessageType</key>
```

```
        <string>Authenticate</string>
        <key>Topic</key>
        <string>com.example.mdm.pushcert</string>
        <key>UDID</key>
        <string> [ redacted ] </string>
    </dict>
</plist>
```

要继续，服务器可以使用空白 plist 作出响应：

```
<?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
    <plist version="1.0">
    <dict>
    </dict>
    </plist>
```

之后包含三个关键的识别信息：接收此响应，客户端向服务器发送“TokenUpdate”请求，

1、PushMagic - MDM 服务器随每个推送请求发送的唯一令牌

2、令牌 - 唯一令牌，用于向 APNS 服务标识设备

3、UnlockToken - 用于清除设备上的密码的托管密钥。

以下列表中所有三个令牌都已被编辑。 PushMagic 标记是一个十六进制字符串，可能是一个简单的随机 UUID(以 UUID 格式)。 令牌是一个 32 字节的二进制值，以 Base64 编码文本表示，UnlockToken 是一个更长的二进制字符串（接近 2 千字节），也是在 Base64 中编码的。

```
PUT: /checkin

        <?xml version="1.0" encoding="UTF-8"?>

        <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"

        "http://www.apple.com/DTDs/PropertyList-1.0.dtd">

        <plist version="1.0">

        <dict>

            <key>MessageType</key>

            <string>TokenUpdate</string>

            <key>PushMagic</key>

            <string> [ redacted uuid string ] </string>

            <key>Token</key>

            <data> [ 32 byte string, base64 encoded, redacted ] </data>

            </data>

            <key>Topic</key>

            <string>com.example.mdm.pushcert</string>

            <key>UDID</key>

            <string> [ redacted ] </string>

            <key>UnlockToken</key>

            <data>

                [ long binary string encoded in base64, redacted ]

            </data>

        </dict>
```

```
    </plist>
```

同样，除了简单的空白 plist 之外，服务器不需要提供任何响应。 完成此操作后，服务器会保留令牌的副本，并且该设备现在已完全注册。

发送推送通知

当 MDM 服务器需要联系设备时，它通过 APNS 向设备发送通知。 推送通知使用特殊格式，其中顶级“aps {}”列表被替换为单个“mdm”值。我们的代码使用了一个开源的 Apns 推送框架，Github 地址是：https://github.com/notnoop/java-apns，推送通知需要设备的 APNS 令牌和 mdm PushMagic 令牌。

使用 APNSWrapper 发送通知非常简单：

```
    IApnsService service = getApnsService(p12Path);
    Payload payload = new Payload();
    payload.addParam("mdm", mdm.getPushMagic());
    service.sendNotification(mdm.getToken(), payload);
```

由此产生的 APNS 通知的有效载荷大致如下所示：

```
{"aps":{},"mdm":"996ac527-9993-4a0a-8528-60b2b3c2f52b"}
```

一旦发送到 APNS 服务器，通知应该快速中继到设备（假设它当前连接到 APNS），然后设备将联系 MDM 服务器以获取进一步说明。

设备响应推送

设备收到推送通知后，会联系服务器以接收指令。 从这里开始，它使用“服务器”URL（而不是在注册期间使用的“签入”URL），但所有内容都通过 HTTP PUT 发送。 首先，设备发送简

单的“状态：空闲”消息：

```
PUT: /server

    <?xml version="1.0" encoding="UTF-8"?>

    <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"

    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">

    <plist version="1.0">

    <dict>

        <key>Status</key>

        <string>Idle</string>

        <key>UDID</key>

        <string> [ redacted ] </string>

    </dict>

    </plist>
```

然后，服务器将响应任何排队等待设备的命令。 在所有情况下，命令响应都包含一个“CommandUUID”字段，该字段应包含唯一的随机 UUID。 这允许服务器匹配从客户端收到的响应发送的命令，但不是必需的（并且可能当前空白）。

大多数命令只是命令名称，但少数命令需要附加参数。 详细信息在《附录 B》的命令列表中提供。举个简单的例子，我们将看两条命令：DeviceLock 和 ClearPasscode。

设备锁定

DeviceLock 命令不需要参数，只是简单的如下：

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
	<key>Command</key>
	<dict>
		<key>RequestType</key>
		<string>DeviceLock</string>
	</dict>
	<key>CommandUUID</key>
	<string></string>
</dict>
</plist>
```

当它被提供给设备时，作为对状态：空闲命令的响应，设备立即锁定。 来自设备的响应是标准确认消息：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
	<key>CommandUUID</key>
	<string></string>
```

```
        <key>Status</key>
        <string>Acknowledged</string>
        <key>UDID</key>
        <string> [ redacted ] </string>
    </dict>
</plist>
```

清除密码

ClearPasscode 命令需要设备的 UnlockToken（在注册阶段在 UpdateToken 消息中提供给服务器）：

```
<?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
    "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
    <plist version="1.0">
    <dict>
        <key>Command</key>
        <dict>
            <key>RequestType</key>
            <string>ClearPasscode</string>
            <key>UnlockToken</key>
            <data>
        [ redacted ]
```

```
                </data>
            </dict>
            <key>CommandUUID</key>
            <string></string>
        </dict>
        </plist>
```

如果 UnlockToken 与注册期间创建的内容相匹配，则设备应清除密码。 它将保持锁定状态，但不需要密码解锁。 如果存在需要密码的策略，则会给用户一个宽限期以输入新的密码。

如果安装了更改密码要求的新策略，该警告也会出现。

结论

总的来说，Apple 的 MDM 协议相当简单，并且不难实现。《附录 A》列出了迄今为止已公开的协议的大部分内容，而《附录 B》则提供了本文中描述的 MDM 测试服务器的源代码。虽然提供的 OpenMDMServer 代码不完善以及可能出现一些缺陷，但希望我的这一份努力能够激发更多的开发者对 iOS 移动设备管理的安全性和可靠性的进一步研究。

附录 A - 命令列表

Control Commands （控制类命令）	• Device Lock（锁屏） • Erase Device（擦除数据） • Clear Passcode （清除密码）
Device Queries （设备查询类命令）	• Security Information （设备安全信息） • Installed Application List （安装的 APP） • Device Information （设备信息） • Certificate List （证书数据） • Profile List （描述文件） • Provisioning Profile List（预植描述文件） • Restrictions List （限制信息）
Device Configuration （设备配置）	• Install Profile （安装描述文件） • Remove Profile（移除描述文件） • Install Provisioning Profile（安装预植描述文件） • Remove Provisioning Profile （移除预植描述文件）
Device to Server Commands （设备-服务器命令）	• Authenticate（认证） • Token Update （更新 Token）

命令格式

所有命令都以 Apple Property List（.plist）文件的形式发送。 每个包含一个名为“CommandUUID”的顶级密钥，其中包含用于唯一标识命令实例的 UUID 字符串以及包含附

加信息的顶级密钥“Command”。

```
<plist version="1.0">
    <dict>
        <key>Command</key>
        <dict>
            <key>RequestType</key>
            <string>[command name]</string>
            [... additional parameters as needed ...]
        </dict>
        <key>CommandUUID</key>
        <string></string>
    </dict>
</plist>
```

下面列出每条命令的简短描述和所需的参数。

设备响应

设备通过简单的确认来响应许多命令：

```
<plist version="1.0">
    <dict>
        <key>CommandUUID</key>
        <string></string>
        <key>Status</key>
```

```
        <string>Acknowledged</string>
        <key>UDID</key>
        <string>[device UUID]</string>
    </dict>
</plist>
```

“状态”字段可能包含“已确认”，“错误”，“CommandFormatError”或“NotNow”（有关错误字段的详细信息，请参阅下文）。

在命令引发更多扩展响应的地方(比如 DeviceInformation 查询),这些响应的细节在下面给出。通常，这些命令添加一个顶级字段（如 InstalledApplicationList), 该字段的值为扩展数据，存储为字符串，字典或其他元素的数组。 例如：

```
<plist version="1.0">
    <dict>
        <key>CommandUUID</key>
        <string></string>
        <key>SecurityInfo</key>
        <dict>
            <key>HardwareEncryptionCaps</key>
            <integer>3</integer>
            <key>PasscodeCompliant</key>
            <true/>
            <key>PasscodeCompliantWithProfiles</key>
            <true/>
```

```
        <key>PasscodePresent</key>
        <false/>
    </dict>
    <key>Status</key>
    <string>Acknowledged</string>
    <key>UDID</key>
    <string>[device UUID]</string>
</dict>
</plist>
```

错误消息

错误消息的一般格式与确认相同，“状态”更改为“错误”，并添加一个附加的数组作为“ErrorChain”：

ErrorCode	(integer) A unique identifying error code
ErrorDomain	(string) Category of error
LocalizedDescription	(string) Error message, translated to a localized language
USEnglishDescription	(string) Standardized version of error message

一个特殊的错误信息是“NotNow”，当一个命令由于设备被密码锁定而不能被请求时（例如请求安全信息或安装配置文件），就会看到该信息。 发生这种情况时，只要设备解锁，设备就会尝试重新与 MDM 服务器连接，以便重试该命令。

用无效或缺少参数发送的命令将返回“CommandFormatError”状态。

设备锁定

立即锁定设备。 如果存在密码，则需要该密码才能解锁设备。

RequestType	DeviceLock

擦除设备

立即擦除设备内存并将其重置为“从工厂清理”状态。 需要连接 iTunes 才能从备份中恢复或配置为新的。

RequestType	EraseDevice

清除密码

如果设备上存在密码，该命令将清除该密码。 如果其他配置控件需要密码，则会为用户设置一个宽限期，以设置新的密码。

RequestType	ClearPasscode
UnlockToken	(data) UnlockToken data, base-64 encoded

安全信息

列出设备的指定安全相关设置，包括硬件加密功能，以及是否存在密码（如果是，则是否符合配置）。 如果存在密码，则必须解锁该设备才能执行该命令。

RequestType	SecurityInfo
Queries	(array of strings): “HardwareEncryptionCaps”，

	“PasscodePresent”， “PasscodeCompliant”， “PasscodeCompliantWithProfiles”

响应基于通用确认响应，并附带一个名为“SecurityInfo”的附加字典：

HardwareEncryptionCaps	integer
PasscodePresent	boolean
PasscodeCompliant	boolean
PasscodeCompliantWithProfiles	boolean

安装的应用程序列表

列出设备上当前安装的所有应用程序。 包括应用程序使用的总体持久性存储，以字节表示，以及应用程序的名称，版本和软件包标识符。 不列出通过越狱方法安装的应用程序。

RequestType	InstalledApplicationList

在“InstalledApplicationList”关键字中的其他响应信息是一个字典项目数组：

BundleSize	integer
DynamicSize	integer
Identifier	string
Name	string
Version	string

设备信息

检索有关设备的指定一般信息，包括 MAC 地址，IMEI，电话号码，软件版本，型号名称和编号，

序列号。

RequestType	DeviceInformation
Queries	(array of strings): "AvailableDeviceCapacity", "BluetoothMAC", "BuildVersion", "CarrierSettingsVersion", "CurrentCarrierNetwork", "CurrentMCC", "CurrentMNC", "DataRoamingEnabled", "DeviceCapacity", "DeviceName", "ICCID", "IMEI", "IsRoaming", "Model", "ModelName", "ModemFirmwareVersion", "OSVersion", "PhoneNumber", "Product", "ProductName", "SIMCarrierNetwork", "SIMMCC", "SIMMNC", "SerialNumber", "UDID", "WiFiMAC", "UDID"

响应是一个名为“QueryResponses”的字典，其中包括上述项目作为键。 将会被忽略的响应（例如，来自 iPod Touch 的 PhoneNumber 字段）将被忽略。 AvailableDeviceCapacity 和 DeviceCapacity 是实数字段，而 DataRomingEnabled 和 IsRoaming 是布尔值。 其余的都以字符串形式返回。

证书列表

列出设备上当前安装的所有证书。

RequestType	CertificateList

响应包含一个“CertificateList”字典值数组：

CommonName	string
Data	base-64 cert information
IsIdentity	boolean

描述文件列表

列出安装在设备上的配置文件。 包括通用名称，是否需要删除密码，是否禁止删除，唯一标识符以及其他类似信息。

RequestType	ProfileList

响应键“ProfileList”包含一个字典项目数组：

HasRemovalPasscode	boolean
IsEncrypted	boolean
PayloadDisplayName	string
PayloadIdentifier	string
PayloadRemovalDisallowed	boolean
PayloadUUID	string
PayloadVersion	integer
SignerCertificates	array of data items, each with base-64 cert info
PayloadContent	array of dicts, each with PayloadDisplayName, PayloadIdentifier,

	PayloadType, and PayloadVersion keys.

预植描述文件列表

列出安装在设备上的配置文件（类似于配置文件列表）。

RequestType	ProvisioningProfileList

响应包括一个“ProvisioningProfileList”键，其中包含一个字典值数组：

ExpiryDate	date
Name	string
UUID	string

限制列表

列出当前对设备有效的限制。 例如，列出禁用的应用程序，是否强制启用备份加密等。

RequestType	RestrictionsList

响应包括“GlobalRestrictions”，它是一个包含限制的详细列表的字典，大部分呈现为布尔值。

确切的内容和结构取决于设备上的限制。

安装配置文件

如果给定.mobileconfig 配置文件（由 IPCU 或其他工具创建）的 base-64 编码，则将配置文件安装到设备上。

RequestType	InstallProfile
Payload	(data) IPCU .mobileconfig file, base-64 encoded

删除配置文件

给定一个有效载荷标识符(通常显示为反向 DNS 标识符 ,如“com.example.cfg.restrictions”) ,从设备中删除该配置文件。

RequestType	RemoveProfile
Identifier	(string) Profile identifier

安装预植描述文件

给定.mobileprovision 配置文件（由 IPCU 或其他工具创建）的 base-64 编码，将配置文件安装到设备上。

RequestType	InstallProvisioningProfile
Payload	(data) IPCU .mobileprovision file, base-64 encoded

删除预植描述文件

根据配置文件的 UUID，该命令将从设备中删除配置文件。

RequestType	RemoveProvisioningProfile
UUID	(string) Provisioning profile UUID

设备认证

这是设别发送的用于启动注册的设备命令。可以由服务器使用 ,以允许或拒绝基于设备的 UDID 的注册。 注 - 不遵循与服务器到设备命令相同的格式。 没有 CommandUUID 字段和 Command 字典结构 - 所有参数都是主属性列表字典中的顶级项目。

MessageType	Authenticate
Topic	(string) Subject Name: User ID on APNS push certificate used by server
UDID	(string) Device UDID

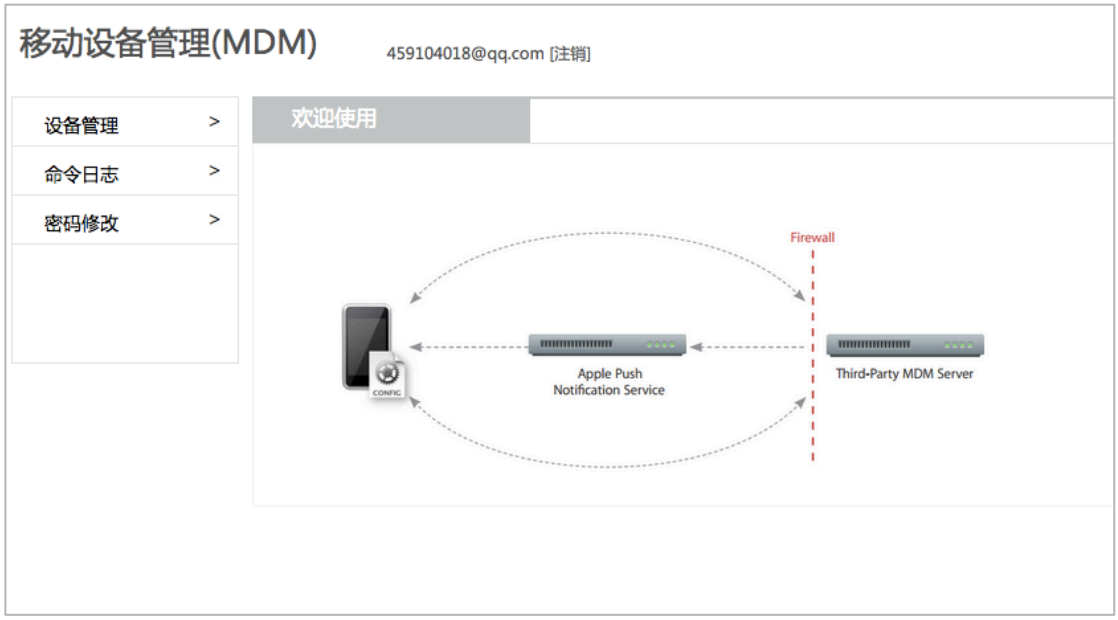
令牌更新

这是设备在注册期间发送的设备消息。 向服务器提供用于通过 APNS 与设备联系的令牌，以及通过 Clear Passcode 命令解锁设备的密钥。 注 - 不遵循与服务器到客户端命令相同的格式。没有CommandUUID字段和Command字典结构 - 所有参数都是主属性列表字典中的顶级项目。

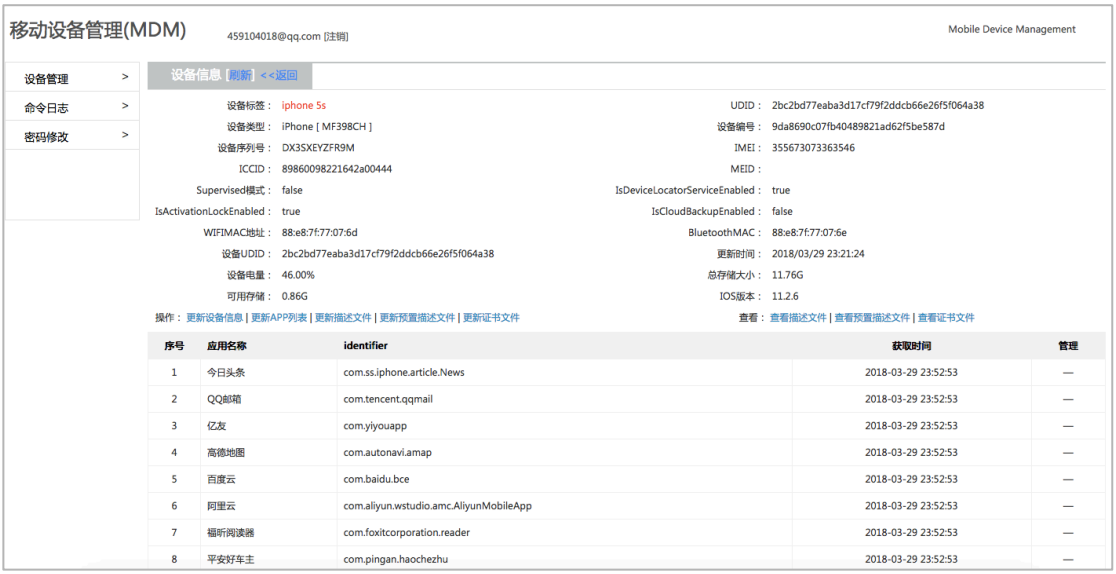
MessageType	Token Update
PushMagic	(string) UUID-like string
Token	(data) 32-byte APNS device token, base-64 encoded
Topic	(string) Subject Name: User ID on APNS push certificate used by server
UDID	(string) Device UDID
UnlockToken	(data) Device unlock key, base-64 encoded

附录 B - 源代码

MDMServer 截图




移动设备管理(MDM)
459104018@qq.com [注销]
Mobile Device Management
设备管理 >
命令日志 >
密码修改 >
设备列表 [刷新]
序号 标签 版本 设备ID 设备类型 设备状态 控制 查看 操作
1 iphone 5s 11.2.6 9da8690c07fb40489821ad62f5be587d iPhone 设备可控 设备锁屏 | 清除密码 | 清除数据 | 安装APP 设备信息 移除
共 1 条数据，页次:1/1 页 首页 上一页 下一页 尾页


移动设备管理(MDM)
459104018@qq.com [注销]
Mobile Device Management
设备管理 >
命令日志 >
密码修改 >
设备信息 [刷新] <<返回
设备标签： iphone 5s
设备类型： iPhone [MF398CH]
设备序列号： DX3SXEYZFR9M
ICCID： 89860098221642a00444
Supervised模式： false
IsActivationLockEnabled： true
WIFIMAC地址： 88:e8:7f:77:07:6d
设备UDID： 2bc2bd77eaba3d17cf79f2ddcb66e26f5f064a38
设备电量： 46.00%
可用存储： 0.86G
UDID： 2bc2bd77eaba3d17cf79f2ddcb66e26f5f064a38
设备编号： 9da8690c07fb40489821ad62f5be587d
IMEI： 355673073363546
MEID：
IsDeviceLocatorServiceEnabled： true
IsCloudBackupEnabled： false
BluetoothMAC： 88:e8:7f:77:07:6e
更新时间： 2018/03/29 23:21:24
总存储大小： 11.76G
IOS版本： 11.2.6
操作： 更新设备信息 | 更新APP列表 | 更新描述文件 | 更新预置描述文件 | 更新证书文件
查看： 查看描述文件 | 查看预置描述文件 | 查看证书文件
序号 应用名称 identifier 获取时间 管理
1 今日头条 com.ss.iphone.article.News 2018-03-29 23:52:53 —
2 QQ邮箱 com.tencent.qqmail 2018-03-29 23:52:53 —
3 亿友 com.yiyouapp 2018-03-29 23:52:53 —
4 高德地图 com.autonavi.amap 2018-03-29 23:52:53 —
5 百度云 com.baidu.bce 2018-03-29 23:52:53 —
6 阿里云 com.aliyun.wstudio.amc.AliyunMobileApp 2018-03-29 23:52:53 —
7 福昕阅读器 com.foxitcorporation.reader 2018-03-29 23:52:53 —
8 平安好车主 com.pingan.haochezhu 2018-03-29 23:52:53 —


移动设备管理(MDM)
459104018@qq.com [注销]
设备管理 >
命令日志 >
密码修改 >
APP信息 [刷新] <<返回
设备类型： iPhone [MF398CH]
设备编号： 9da8690c07fb40489821ad62f5be587d
安装类型： APP ID PList安装
输入值：
提交请求

代码地址

1、演示站：http://mdm.mbaike.net

2、GitHub 开源地址：https://github.com/keaijohnee/OpenMDMServer

3、OSChina 开源地址：http://git.oschina.net/jianggege/OpenMDMServer

技术咨询及服务

咨询 QQ：459104018　　微信：13568933413（电话同号）　　QQ 群：205891305；

附录 C - 参考文献

1、 http://www.mbaike.net/mdm/6.html　基于 IOS 上 MDM 技术相关资料整理及汇总

2、 https://mdm.mbaike.net　Apple iOS 移动设备管理

3、 https://github.com/notnoop/java-apns/　Java APNS 推送

4、 http://www.rootmanager.com/iphone-ota-configuration/iphone-ota-setup-with-signed-mobileconfig.html　mobileconfig 配置文件的签名和认证

5、 https://github.com/keaijohnee/OpenMDMServer　GitHub 开源地址

6、 http://git.oschina.net/jianggege/OpenMDMServer　OSChina 开源地址

2018 年 3 月 31 日

江哥一直在