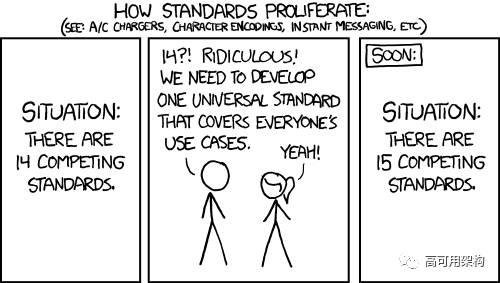
Wolfram Hempel 是 deepstreamIO 的聯合創始人。deepstreamIO 是一家位於德國的技術創業公司，為移動客戶端、及物聯網設備提供高性能、安全和可擴展的實時通信服務。文本由魏佳翻譯，轉載譯文請註明來自高可用架構。

想要向伺服器發送請求並獲得響應？直接使用 HTTP 吧！非常簡單。但是當需要通過持久的雙向連接來通信時，如 WebSockets，當然你也有其它的選擇。

這篇文章會簡單扼要的解釋 MQTT，XMPP，STOMP，AMQP，WAMP 和其它替代品。這裡常被引用的 XKCD 漫畫[1]之一：



**等等，其實沒有「實時協議」這種東西！**

確實如此，但這裡我將該術語（Realtime Protocol）當作一堆協議的代稱，這類協議都是設計用於分發消息，同步數據和通過持久雙向的連接來進行請求/響應。

我們先來根據它們各自的目的進行分類：

**純消息**

底層協議（例如 TCP）是被設計用來將一個消息從一個發送者（sender）傳遞給一個接收者（receiver）。他們並不關係消息本身應該如何構建，也不關係消息的請求、獲取、存儲以及如何保證安全可靠。

像 WebSockets 這樣在 TCP 之上的協議，添加了一些額外的功能，例如使用頭部（header）傳輸元數據，通過多個數據包分割較大的消息，簡單的身份驗證，以及路由／重定向相關信息。本質上它們仍然是點對點交換數據的方式。

當涉及到構建更大，更複雜的系統時，你需要一個更高層次的通信範式：

**發布－訂閱**

發布－訂閱模式是在大規模系統中被廣泛使用的通信方式，用於多對多無狀態消息傳遞。「訂閱者」（Subscribers）可以訂閱「消息主題」（topics，channels，events，namespaces）， 「發布者」（Publishers）可以將消息發布到「消息主題」，通過路由，所有的訂閱者都將收到。

這種範式是非常靈活，高效和可擴展。它將發送者與接收者隔離開，允許訂閱者自由得訂閱主題或取消訂閱。這和我們日常訂閱報紙是一樣的。

有許多支持發布－訂閱的協議：MQTT，STOMP，WAMP 等等。那麼我們應該如何選擇呢？



**MQTT**

MQTT（Message Queue Telemerty Transport）[2]是一種二進位協議，主要用於伺服器和那些低功耗的物聯網設備（IoT）之間的通信。

它位於 TCP 協議的上層，除了提供發布－訂閱這一基本功能外，也提供一些其它特性：不同的消息投遞保障（delivery guarantee），「至少一次」和「最多一次」。通過存儲最後一個被確認接受的消息來實現重連後的消息恢復。

它非常輕量級，並且從設計和實現層面都適合用於不穩定的網絡環境中。

**什麼時候應使用它？**

物聯網（IoT）場景中更適合，支持幾乎所有語言進行開發，並且瀏覽器也可通過 WebSocket 來發送和接收 MQTT 消息。

同時，對於MQTT Broker，也有很多選擇，如 Mosquitto[3] 或 VerneMQ[4] 以及基於雲的 MQTT 平台，如 HiveMQ[5] 或 CloudMQTT[6]。



**STOMP**

面向流文本的消息傳輸協議（STOMP，Streaming Text Oriented Messaging Protocol）[7]，是 WebSocket 通信標準。在通常的發布－訂閱語義之上，它通過 begin/publish/commit 序列以及 acknowledgement 機制來提供消息可靠投遞。

由於協議簡單且易於實現，幾乎所有的程式語言都有 STOMP 的客戶端實現。但是在消息大小和處理速度方面並無優勢。

**什麼時候會使用它？**

當與 Apache Apollo[8] 這樣的多協議代理（multi-protocol broker）中間件結合使用時，可以做許多有趣的集成。比如在瀏覽器的圖表中不斷更新 IoT 設備的數據。

**選擇二進位還是基於文本？**

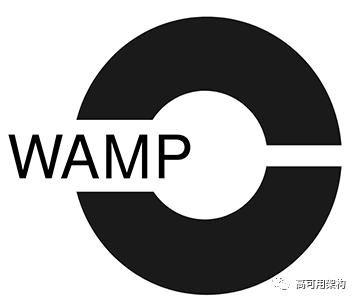
到目前為止，我們已經講了兩個協議：一個二進位、另一個基於文本。讓我們快速比較一下差異：

通過控制線纜中光或電的打開或關閉（邏輯開關），或控制 WiFi 信號的波峰或波谷來實現計算機之間的信息交換。從原理上來說，這是基於二進位的形式。因此，從這個層面來說所有協議都是二進位協議。

信息一旦發送，接收方有兩個選擇：它可以將 0/1 流分組成字節序列，進而獲取（解析）信息；或者可以執行額外的步驟，將其轉換為文本，然後再解析此文本。

前一種方法稱為（基於）二進位的。它節省了一些昂貴的操作，同時是傳輸任何非文本信息的標準形式。例如，圖像，音頻，視頻或文件。當然它也可用於發送文本信息。例如，通過向每個消息增加幾個字節來表達元信息，比如描述該消息的長度或類型，這樣就只需將實際的消息數據轉換為文本。

然而，由於在許多發布－訂閱式的架構中，信息交換是基於文本的，所以許多協議選擇簡單地將整個信息轉化為文本，從而降低複雜性並提高了可讀性，當然帶來的代價就是需要再消息接受後執行額外的計算任務。



**WAMP**

Web 應用消息協議（WAMP，Web Application Messaging Protocol）[9]，它嘗試開發一種開放的、基於文本的協議標準，並且結合了基於發布－訂閱的請求/響應編程模型，同時具備強大的路由和消息投遞策略。目前它被廣泛用於集成 crossbar.io[10] 路由器和 autobahn 的高速緩存客戶端[11]。

**Pusher / PubNub＆Co**

那些實時通信平台即服務（Realtime platform-as-a-service）的產品，例如 Pusher 或 PubNub，通常使用它們自己的專有協議。Pusher 已經公開了它們研發的基於 JSON 協議的詳細規範[12]，並且鼓勵第三方或社區幫助構建不同語言的客戶端。Pubnub 雖然更封閉一些，但它們目前支持一系列其它開放協議進行交互，如 MQTT。

**隊列**

有些場景下，簡單的發布－訂閱模式還不夠。這就是隊列存在的場合：它支持多種不同的消息和存儲的模式。

通過獲取／確認策略，消費者接收到隊列的一些消息，確認他們的「消費」，處理它們，然後取下一批消息。這是一個強大同時常用的方式。

想像一下，你正在構建一個類似 Instagram 的產品，它允許用戶上傳圖片並添加各種濾鏡。應用濾鏡的過程是一個計算密集型的操作。因此，不能在上傳時直接進行操作，所以接收圖像的伺服器只是記錄下在文件系統中的位置，接著將「什麼圖像需要使用什麼樣的濾鏡」這個任務添加到任務隊列中。

另一台專門用於處理濾鏡的機器可以從任務隊列中獲取這個任務，讀取原始圖像文件，應用濾鏡，並確認這個任務已經消費（完成）。之後為了水平擴展圖像處理的能力，只需要添加更多的消費者（處理濾鏡的機器）來處理任務隊列即可。

這種模式非常易於擴展，可以添加複雜的路由策略，任務分配策略等。



**AMQP**

高級消息隊列協議（AMQP，Advanced Message Queuing Protocol）[13]是各種消息隊列協議中的佼佼者。RabbitMQ[14] 和 HornetQ[15] 都是實現該協議的流行中間件。

**什麼時候會使用它？**

當簡單的發布－訂閱模型不能滿足使用要求。AMQP 十分可靠且功能強大。當然它及它的實現並不是足夠輕量級。如果你需要一個更輕量級的選擇，那接下來的內容可能會更好：



**ZeroMQ**

ZeroMQ[16] 既是一個協議，也是一套協議實現的組件。提供比 AMQP 更高速同時去中心化的替代方案。

**什麼時候會使用它？**

當你需要海量吞吐以及無單點故障風險的消息隊列支撐你的複雜工作流，那麼 ZeroMQ 是個不錯的選擇，當然，你需要對陡峭的學習曲線做好準備。



**JMS**

Java 消息服務（JMS，Java Messaging Service）[17]，是協議同時也是 Java 消息服務規範的標準實現，同時也是 Java 企業版（JEE）規範的一部分。

**什麼時候應使用它？**

當你使用 Java 棧，同時也為 Java Enterprise Platform 付費了，那 JMS 是最佳選擇。如果沒有，那就優先考慮上述那些方案。

**請求/響應**

有時我們只需要發送單個請求，並等待收到一個響應，這完全可以使用HTTP請求完成 ／ 。 但是既然你已經建立了一個與伺服器的持久連接 ，那為什麼不利用它呢？

這種通過持久連接進行的請求／響應模式的通信過程，通常被稱為遠程過程調用（RPC，Remote Procedure Calls）或遠程方法調用（RMI，Remote Method Invocation）。AMQP 或 ZeroMQ 可以通過響應隊列（response-queue）來實現這種模式，JMS 可以直接支持 Java RMI。

**DataSync**

DataSync 是實現實時通信的最新可選方案。

DataSync 將數據存儲中的數據同步給客戶端。客戶端對數據的變更都將同步給所有的訂閱者。DataSync 隱藏了實時通信類應用中維護數據狀態的細節，降低了複雜性，並極大加快了開發速度，但它目前仍然是一種沒有開放的協議標準。



目前 DataSync 已可以在幾個 PaaS 平台上使用，如 deepstreamHub[18]，Firebase[19] 或 Realm[20]。

**總結一下**

利用 deepstream[21] 的分布式實時協議（DRP，Distributed Realtime Protocol），我們有信心找到一種方法，將所有上述概念組合成一個協議，同時確保消息大小，可擴展性和互操作性方面的效率最大化。

**參考**

1. https://xkcd.com/
2. http://mqtt.org/
3. https://mosquitto.org/
4. https://vernemq.com/
5. http://www.hivemq.com/
6. https://www.cloudmqtt.com/
7. https://stomp.github.io/
8. https://activemq.apache.org/apollo/
9. http://wamp-proto.org/
10. http://crossbar.io/
11. http://crossbar.io/autobahn/
12. https://pusher.com/docs/pusher\_protocol
13. https://www.amqp.org/
14. https://www.rabbitmq.com/
15. http://hornetq.jboss.org/
16. http://zeromq.org/
17. http://docs.oracle.com/javaee/6/tutorial/doc/bncdq.html
18. https://deepstreamhub.com/
19. https://firebase.google.com/
20. https://realm.io/
21. https://deepstream.io/

原文網址：<https://kknews.cc/tech/nxj8493.html>