**Chương 2: HTTP You are the Client and you are the boss**

HTTP (HyperText Tranfer Protocol) là “ngôn ngữ” trong đó client và server của một ứng dụng Web có thể tương tác với nhau. Nó được định nghĩa vào năm 1996, tính đơn giản và tính linh hoạt của nó ở mức độ nào đó, chịu trách nhiệm cho sự thành công và mở rộng của các trang web và toàn thể Internet.

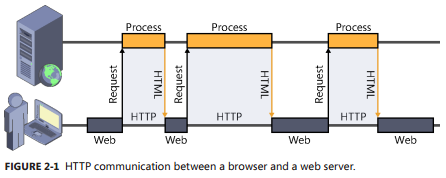
Mặc dù nó vẫn có giá trị trong các kịch bản web truyền thống, có những điểm khác, chẳng hạn như các ứng dụng thời gian thực hoặc các dịch vụ, mà nó là khá hạn chế.

**Hoạt động của HTTP**

Một hoạt động của HTTP dựa trên một giản đồ phản hồi yêu cầu, luôn được khởi tạo bởi client. Thủ tục này thường được gọi là mô hình pull: Khi client cần truy cập vào một tài nguyên được lưu trữ bởi một server, nó sẽ bắt đầu kết nối với nó và yêu cầu thông tin mong muốn sử dụng "ngôn ngữ" được định nghĩa bởi giao thức HTTP. Server xử lý yêu cầu này, trả về tài nguyên đã được yêu cầu (có thể là nội dung của một tệp hiện có hoặc kết quả của quá trình chạy) và kết nối ngay lập tức đóng lại.

Nếu client cần có một tài nguyên mới, quá trình bắt đầu lại từ đầu: một kết nối đến server được mở ra, yêu cầu cho tài nguyên được gửi, server xử lý nó, nó trả về kết quả, và sau đó kết nối được đóng lại. Điều này xảy ra mỗi khi chúng ta truy cập trang web, hình ảnh hoặc tài nguyên khác do trình duyệt tải xuống, để đặt tên cho một vài ví dụ.

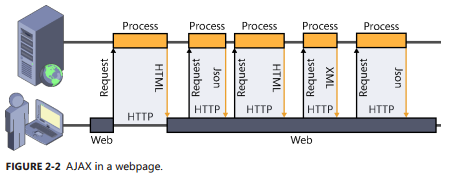
Như bạn có thể đoán bằng cách nhìn vào hình 2-1, nó là một quá trình đồng bộ: sau khi gửi yêu cầu đến server, client là để chờ đợi, không làm gì cho đến khi đáp ứng có sẵn.



Mặc dù hoạt động này là một hệ thống web cổ điển nhưng bản thân giao thức HTTP có thể hỗ trợ các yêu cầu cho sự không đồng bộ của các ứng dụng hiện đại do các kỹ thuật thường được gọi là AJAX (Asynchronous JavaScript and XML).

Sử dụng kỹ thuật AJAX, việc trao đổi thông tin giữa client và server có thể được thực hiện mà không cần rời khỏi trang hiện tại. Tại bất kỳ thời điểm nào, như thể hiện trong hình 2-2, client có thể bắt đầu một kết nối đến máy chủ bằng cách sử dụng JavaScript, yêu cầu một tài nguyên và xử lý nó (ví dụ, cập nhật một phần của trang).

Điều thực sự thuận lợi và đã góp phần vào sự nổi lên của các dịch vụ rất năng động và tương tác, chẳng hạn như Facebook hay Gmail, là các hoạt động này được thực hiện không đồng bộ - có nghĩa là người dùng có thể tiếp tục sử dụng hệ thống trong khi đó liên lạc với server để gửi hoặc nhận thông tin.

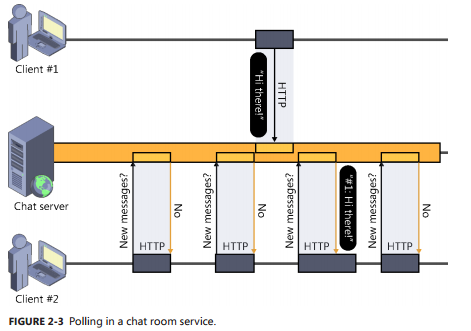


Sơ đồ hoạt động này tiếp tục sử dụng và tuân thhủ giao thức HTTP và mô hình phản hồi yêu cầu của client. Client luôn luôn là chủ động, quyết định khi nào kết nối với server. Tuy nhiên, có những tính huống trong đó HTTP không phải là rất hiệu quả. Với giao thức này, không dễ thực hiện các ứng dụng nhắn tin nhanh hoặc phòng chat, công cụ cộng tác, trò chơi trực tuyến nhiều người hoặc các dịch vụ thông tin thời gian thực, ngay cả khi sử dụng không đồng bộ.

Lý do rất đơn giản: HTTP không được định hướng theo thời gian thực. Có các giao thức khác, chẳng hạn như IRC phổ biến, thực sự tập trung vào việc đạt được giao tiếp nhanh hơn để cung cấp các dịch vụ năng động và tương tác hơn so với những gì chúng ta có thể thu được bằng cách sử dụng pull. Trong đó, server có thể chủ động và gửi thông tin cho client bất cứ lúc nào mà không cần chờ client yêu cầu nó một cách rõ ràng.

**Polling: Câu trả lời?**

Là một nhà phát triển web, khi chúng ta phải đối mặt với một tình huống mà trong đó chúng ta cần server là một trong những thông tin gửi tới client một cách tự động tự chủ, giải pháp đầu tiên mà trực quan hóa đến suy nghĩ của chúng ta là sử dụng kỹ thuật được gọi là **Polling**. **Polling** về cơ bản bao gồm việc tạo các kết nối định kỳ từ client để kiểm tra xem có bất kỳ cập nhật nào có liên quan tại server như thể hiện trong hình 2-3.



Các ưu điểm chính của giải pháp này là: Trước tiên, việc triển khai dễ dàng và thứ hai là ứng dụng phổ cập của nó: nó hoạt động trong mọi trường hợp, với tất cả các trình duyệt và với tất cả các server, bởi vì nó không có gì hơn là sử dụng các tính năng tiêu chuẩn của HTTP. Và tất nhiên, chúng ta vẫn sử dụng mô hình pull.

Tuy nhiên, đôi khi giá Polling quá cao. Kết nối liên tục và ngắt kết nối có chi phí cao về băng thông và xử lý ở cả hai đầu của truyền thông. Phần tồi tệ nhất là chi phí này tăng tương ứng với nhu cầu cập nhật nhanh hơn và số lượng client sử dụng dịch vụ tại một thời điểm nhất định. Trong ứng dụng cung cấp cập nhật thời gian thực, bạn dễ dàng tưởng tượng được tải mà server phải chịu khi có hàng nghìn người dùng kết nối, yêu cầu nhiều bản cập nhật mỗi giây.

Có những kỹ thuật để giảm thiểu các vấn đề này càng nhiều càng tốt. Một trong số đó là sử dụng khoảng thời gian thích nghi để khoảng thời gian giữa các truy vấn thường xuyên thích nghi với tải hệ thống hiện tại hoặc để xác suất cập nhật mới. Giải pháp này khá dễ thực hiện và có thể cải thiện đáng kể tiêu dùng tài nguyên trong một số tính huống.

Có một biến thể bảo thủ hơn về polling, nhưng nó làm giảm trải nghiệm người dùng. Đó là kỹ thuật được gọi là “piggy backing”, bao gồm không phải là đưa ra các câu hỏi cố ý từ client và thay vào đó, lợi dụng bất kỳ sự tương tác giữa người sử dụng và hệ thống để cập nhật bất kỳ thông tin cần thiết. Để minh hoạ điều này, hãy xem xét một dịch vụ web mail: thay vì tạo các truy vấn định kỳ để kiểm tra sự xuất hiện của tin nhắn mới, những kiểm tra sẽ được thực hiện mỗi khi người dùng truy cập vào một trang, Email, hoặc bất kỳ tính năng nào khác. Điều này có thể hữu ích trong các kịch bản không đòi hỏi sự “immediacy” và trong đó các tính năng của hệ thống chính nó có nghĩa là chúng ta có thể chắc chắn rằng người dùng sẽ tương tác với ứng dụng thường xuyên.

Tất nhiên, các biến thể này có thể được kết hợp với nhau để đạt được hiệu quả sử dụng hơn nguồn lực, cung cấp đồng thời một kinh nghiệm người dùng sử dụng hợp lý. Ví dụ, để có được cập nhật, nó sẽ có thể cập nhật tình trạng của một client thông qua “piggy backing”, khi client tương tác với server, sử dụng Polling có hoặc không có sự tuần hoàn thích ứng khi không có sự tương tác.

Kết luận, Polling là một lựa chọn hợp lý mặc dù bất lợi của nó khi chúng ta muốn có một giải pháp đó là dễ thực hiện và có thể được sử dụng phổ quát và trong các tình huống trong đó một tần số rất cao cập nhật là không cần thiết. Trên thực tế, nó được sử dụng rất nhiều trong các hệ thống hiện tại. Một ví dụ thực tế của nó ứng dụng được tìm thấy trong phiên bản web của Twitter, nơi polling được sử dụng để cập nhật thời gian biểu mỗi 30 giây.

**Push: Server chủ động**

Có những ứng dụng mà việc sử dụng pull không phải là rất hiệu quả. Trong số đó, chúng ta có thể đặt tên hệ thống nhắn tin tức thì, bộ công cụ cộng tác thời gian thực, trò chơi trực tuyến người dùng đa người, dịch vụ phát thanh thông tin và bất kỳ loại hệ thống nào cần gửi thông tin tới client đúng khi nó được tạo ra.

Đối với các ứng dụng như vậy, chúng ta cần server chủ động và có khả năng gửi thông tin cho client một cách chính xác khi một sự kiện có liên quan xảy ra, thay vì chờ đợi client yêu cầu.

Đây chính là ý tưởng đằng sau khái niệm Push, **Server chủ động** . Tên này không tham chiếu đến một thành phần, một công nghệ hoặc một giao thức: nó là một khái niệm, một mô hình truyền thông giữa client và server, nơi mà nó là một trong những chủ động liên lạc.

Khái niệm này không phải là mới. Có những giao thức được đẩy theo khái niệm, chẳng hạn như IRC, giao thức điều khiển hoạt động của các dịch vụ phòng chat cổ điển, hoặc SMTP, giao thức phụ trách điều phối việc gửi email. Những điều này đã được tạo ra trước khi thuật ngữ xác định kiểu giao tiếp này đã được đặt ra.

Để server có thể thông báo sự kiện theo thời gian thực cho một nhóm client quan tâm đến việc tiếp nhận chúng, tình huống lý tưởng sẽ có khả năng bắt đầu kết nối điểm-điểm trực tiếp với họ. Ví dụ, một server phòng chat sẽ giữ một danh sách với các địa chỉ IP của các client kết nối và mở một kết nối kiểu socket cho mỗi client để thông báo về sự xuất hiện của một tin nhắn mới.

Tuy nhiên, đó là kỹ thuật không thể. Vì lý do bảo mật, thường không thể thực hiện kết nối trực tiếp tới máy tính client vì có nhiều cấp độ trung gian sẽ từ chối nó, chẳng hạn như tường lửa, đường đi hoặc proxy. Vì lý do này, thực tiễn tập quán là cho client là bắt đầu kết nối chứ không phải ngược lại.

Để phá vỡ vấn đề này và quản lý để có được một hiệu quả tương tự, một số kỹ thuật xuất hiện được dựa trên các yếu tố tích cực được nhúng trong các trang web (Java applet, Flash, ứng dụng Silverlight, v.v.). Các thành phần này thường được sử dụng ổ cắm để mở một kết nối liên tục đến server- tức là một kết nối sẽ vẫn mở cho đến khi client kết nối với dịch vụ, lắng nghe bất cứ thứ gì mà máy chủ phải thông báo. Khi các sự kiện xảy ra liên quan đến client được kết nối, server sẽ sử dụng kênh mở này để gửi cập nhật theo thời gian thực.

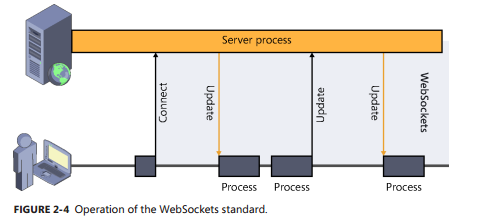
Mặc dù cách tiếp cận này đã được sử dụng trong nhiều giải pháp Push, nó có xu hướng biến mất. Các thành phần hoạt động được nhúng trong các trang đang được loại bỏ khỏi web với tốc độ nhanh chóng và đang được thay thế cho các giải pháp thay thế hiện đại, đáng tin cậy và phổ quát hơn như HTML5. Hơn nữa, các kết nối liên tục lâu dài dựa trên các ổ cắm tinh khiết là vấn đề khi có người trung gian.

Các yếu tố (tường lửa, proxy, v.v ...) có thể chặn các thông tin liên lạc này hoặc đóng các kết nối sau một khoảng thời gian không hoạt động. Chúng cũng có thể gây nguy hiểm cho server. Do nhu cầu về các giải pháp đáng tin cậy để đề cập đến các loại kịch bản này, cả W3C và IETF - các tổ chức chính thúc đẩy và xác định các giao thức, ngôn ngữ và tiêu chuẩn cho Internet - bắt đầu làm việc trên hai tiêu chuẩn cho phép truyền thông trực tiếp và thông thạo từ server cho client. Chúng được biết đến như là WebSockets và Server-Sent Events, và cả hai đều nằm dưới ô dù của tên thương mại "HTML5".

**WebSockets**

Chuẩn WebSockets bao gồm một API phát triển, được xác định bởi W3C (World Wide Web Consortium, http://www.w3.org), và một giao thức truyền thông, trong đó IETF (Internet Engineering Task Force, http: //www.ietf.org) đang làm việc.

Về cơ bản, nó cho phép thiết lập một kết nối liên tục mà client sẽ bắt đầu bất cứ khi nào cần thiết và sẽ vẫn mở. Một kênh hai chiều giữa client và server được tạo ra, nơi mà có thể gửi thông tin đến đầu kia vào bất kỳ lúc nào, như thể hiện trong hình 2-4.



Mặc dù vào thời điểm đó, các yêu cầu kỹ thuật của cả API và giao thức đều khá cao, chúng ta vẫn chưa thể công nhận công nghệ này được áp dụng rộng rãi.

Chúng ta có thể tìm thấy các hiện thực của WebSockets trong nhiều trình duyệt hiện tại, chẳng hạn như Internet Explorer 10, Internet Explorer 11, Chrome và Firefox. Một số tính năng chỉ triển khai một phần (Opera mini, trình duyệt Android) và ở những trình duyệt khác, WebSockets đơn giản không có sẵn.

Bên cạnh vấn đề các cấp thực hiện khác nhau ở phía client, thực tế là tiêu chuẩn bao gồm một giao thức độc lập cho truyền thông (mặc dù ban đầu được thương lượng trên HTTP) có nghĩa là thay đổi cũng phải được thực hiện trên một số yếu tố cơ sở hạ tầng, và thậm chí trên các server, Để kết nối bằng WebSockets được chấp nhận.

Ví dụ, không thể sử dụng WebSockets một cách dễ dàng trên các công nghệ của Microsoft cho đến khi làn sóng phát triển mới nhất (Internet Explorer 10, ASP.NET 4.5, WCF, IIS 8, v.v ...), trong đó nó đã bắt đầu Được hỗ trợ natively.

Từ quan điểm của một nhà phát triển, WebSockets cung cấp một API JavaScript thực sự đơn giản và trực quan để bắt đầu các kết nối, gửi tin nhắn, và đóng các kết nối khi không cần thiết nữa, cũng như các sự kiện để nắm bắt các tin nhắn nhận được:



Như bạn thấy, kết nối được mở một cách đơn giản bằng cách khởi tạo một đối tượng WebSockets trỏ đến URL của điểm cuối dịch vụ. URL sử dụng giao thức ws:// để chỉ ra rằng nó là một kết nối WebSockets.

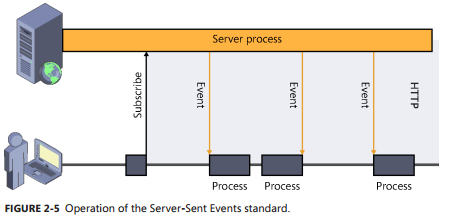
Bạn cũng có thể thấy chúng ta dễ dàng nắm bắt các sự kiện được tạo ra khi chúng ta thành công trong việc mở kết nối, dữ liệu được nhận, hoặc kết nối bị đóng.

Không nghi ngờ gì, WebSockets là công nghệ của tương lai để thực hiện dịch vụ push thời gian thực.

**Server-Sent Event (API Event Source)**

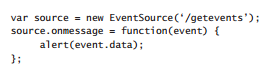
Các Sự kiện gửi bởi server, còn được gọi là Nguồn sự kiện API, là tiêu chuẩn thứ hai mà W3C đã làm việc. Hiện tại, tiêu chuẩn này đang trong trạng thái đề xuất của nhà nước. Nhưng lần này, bởi vì đó là một API JavaScript tương đối đơn giản và không cần thay đổi đối với các giao thức cơ bản, việc triển khai và áp dụng nó đơn giản hơn trong trường hợp của chuẩn WebSockets.

Ngược với sau này, Server-Sent Events đề xuất việc tạo ra một kênh một chiều từ server đến client, nhưng được mở bởi client. Nghĩa là client "đăng ký" vào một nguồn sự kiện có sẵn tại server và nhận các thông báo khi dữ liệu được gửi qua kênh, như được minh họa trong Hình 2-5.



Tất cả các giao tiếp được thực hiện trên HTTP. Sự khác biệt duy nhất liên quan đến kết nối truyền thống là việc sử dụng dòng văn bản/sự kiện-dòng-nội dung trong phản hồi, cho biết rằng kết nối sẽ được giữ nguyên bởi vì nó sẽ được sử dụng để gửi một chuỗi các sự kiện liên tục - Hoặc tin nhắn-từ server.

Thực hiện tại client thậm chí còn đơn giản hơn so với ứng dụng mà chúng ta đã thấy ở WebSockets:



Như bạn có thể đoán, instantiating các đối tượng EventSource bắt đầu đăng ký của client cho dịch vụ mà URL được cung cấp trong contructor, và các tin nhắn sẽ được xử lý trong chức năng gọi lại được chỉ định cho rằng có hiệu lực.

Hiện nay, hầu hết các trình duyệt hỗ trợ tiêu chuẩn này ngoại trừ Internet Explorer và một số trình duyệt di động, và điều này hạn chế việc sử dụng nó trong các ứng dụng thực tế. Ngoài ra, nếu chúng ta nhìn vào nó từ quan điểm cơ sở hạ tầng, chúng ta thấy rằng mặc dù dựa trên HTTP rất đơn giản hóa việc tổng quát của nó, nó đòi hỏi sự trợ giúp của các proxy hoặc các loại trung gian khác, nó phải có khả năng giải thích các loại nội dung được sử dụng và không xử lý các kết nối giống như cách truyền thống - ví dụ như tránh phản ứng đệm hoặc ngắt kết nối do thời gian chờ.

Cũng cần nhấn mạnh những hạn chế do thực tế là kênh được thiết lập cho giao thức này là một chiều từ server đến client: nếu client cần gửi dữ liệu đến server, nó phải thực hiện điều đó thông qua một kết nối khác, thường là một yêu cầu HTTP khác, ví dụ như có mức tiêu thụ tài nguyên lớn hơn nếu WebSockets được sử dụng trong cùng một kịch bản này.

**Pus today**

Như chúng ta đã thấy, tiêu chuẩn và trình duyệt đều đang chuẩn bị để giải quyết các kịch bản push cổ điển, mặc dù hiện tại chúng ta không có đủ an ninh để sử dụng chúng một cách phổ quát.

Tuy nhiên, đẩy là cái gì đó mà chúng ta cần ngay bây giờ. Người dùng yêu cầu các ứng dụng tương tác, nhanh nhẹn hơn và hợp tác hơn. Để phát triển chúng, chúng ta phải sử dụng các kỹ thuật cho phép chúng ta đạt được sự thúc đẩy ngay tức khắc nhưng phải tính đến những hạn chế hiện tại trong các trình duyệt và cơ sở hạ tầng. Tại thời điểm này, chúng ta có thể có được điều đó chỉ bằng cách tận dụng lợi thế của HTTP và sự phổ biến của nó.

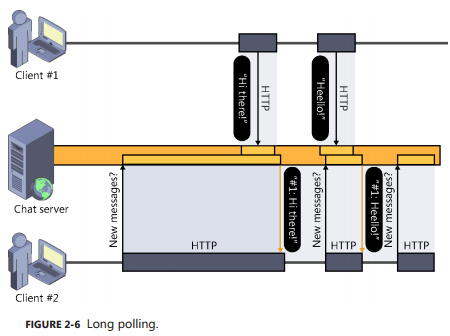
Với các cơ sở này, rất dễ dàng tìm thấy nhiều đề xuất khái niệm trên Internet, chẳng hạn như Comet, push HTTP, reverse AJAX, AJAX push, và như vậy, mỗi mô tả các giải pháp (đôi khi coinciding) để đạt được các mục tiêu mong muốn. Trong cùng một cách, chúng ta có thể tìm thấy các kỹ thuật cụ thể khác nhau mô tả làm thế nào để thực hiện đẩy trên HTTP ít nhiều hiệu quả, chẳng hạn như “long polling”, XHR streaming, hoặc oforever frame.

Bây giờ chúng ta sẽ nghiên cứu hai trong số đó, cuộc thăm dò lâu dài và khung vĩnh cửu, vì hai lý do chính. Thứ nhất, bởi vì chúng là những hệ thống phổ quát nhất (chúng hoạt động trong tất cả các loại client và hệ thống server), và thứ hai, bởi vì chúng được sử dụng bởi SignalR, như chúng ta sẽ thấy sau này. Vì vậy, sẽ hướng tới mục tiêu của cuốn sách này.

**Long Polling**

Kỹ thuật Push này khá giống với polling đã mô tả nhưng nó giới thiệu những sửa đổi nhất định để nâng cao hiệu quả truyền thông và tính trực tiếp.

Trong trường hợp này, client cũng polling cho các bản cập nhật, nhưng, không giống như trong polling, nếu không có dữ liệu nào đang chờ nhận, kết nối sẽ không tự động đóng lại và được khởi tạo lại sau. Trong Long Polling, kết nối vẫn mở cho đến khi server có thông báo gì đó, như trong hình 2-6.



Kết nối, luôn được khởi xướng bởi client, có thể được đóng lại vì chỉ có hai điều:

* Máy chủ gửi dữ liệu tới client thông qua kết nối.
* Lỗi thời gian chờ xảy ra do thiếu hoạt động về kết nối.

Trong cả hai trường hợp, một kết nối mới sẽ được thiết lập ngay lập tức, mà sẽ vẫn chờ đợi bản cập nhật.

Kết nối này được sử dụng độc quyền để nhận dữ liệu từ server, vì vậy nếu client cần gửi thông tin trở lên, nó sẽ mở một kết nối HTTP song song được sử dụng riêng cho mục đích đó.

Ưu điểm chính của việc “long polling” là độ trễ thấp trong việc cập nhật client, bởi vì ngay khi server có dữ liệu để cập nhật trạng thái của client, nó sẽ được gửi qua kênh đã mở, do đó đầu kia sẽ nhận được trong thời gian thực.

Ngoài ra, vì số lượng các lỗ nối và đóng cửa đã được giảm, tối ưu hóa tài nguyên ở cả hai đầu là cao hơn nhiều so với polling.

Hiện nay, đây là một giải pháp được sử dụng rộng rãi do thực hiện tương đối đơn giản của nó và thực tế là nó là hoàn toàn phổ quát. Không có tính năng trình duyệt cụ thể nào được sử dụng-chỉ khả năng được cung cấp bởi HTTP.

Tiêu thụ tài nguyên với thời gian “long polling” là hơi cao hơn so với các kỹ thuật khác, nơi kết nối được giữ mở. Lý do là vẫn có nhiều kết nối mở và đóng nếu tỷ lệ cập nhật cao, không quên các kết nối bổ sung đã được sử dụng khi client muốn gửi dữ liệu đến server. Đồng thời, thời gian cần thiết để thiết lập kết nối có nghĩa là có thể có sự chậm trễ giữa các thông báo. Những sự chậm trễ này có thể trở nên rõ ràng hơn nếu các server đã gửi một loạt các thông báo kế tiếp cho client. Trừ khi chúng ta triển khai một số loại tối ưu hóa, chẳng hạn như đóng gói nhiều tin nhắn vào cùng một phản hồi HTTP, mỗi tin nhắn sẽ phải chờ để được gửi trong khi client nhận được thông báo trước đó trong chuỗi, xử lý và mở lại kênh để yêu cầu cập nhật.

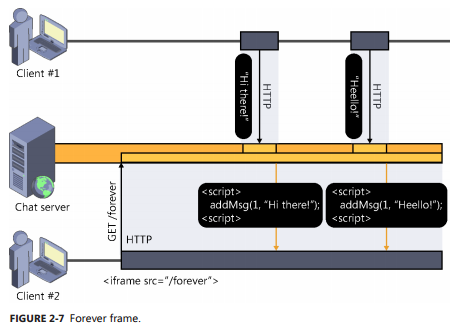
**Forever frame**

Kỹ thuật khác mà chúng ta sẽ nhìn vào được gọi là **Forever frame** và sử dụng thẻ HTML <IFRAME> một cách khéo léo để có được một kết nối mở vĩnh viễn. Theo một cách nào đó, điều này rất giống với các sự kiện đã gửi của server.

Nói chung, nó bao gồm nhập một thẻ <IFRAME> trong trang đánh dấu của client. Trong nguồn của <IFRAME>, URL mà máy chủ đang nghe được xác định. Server sẽ duy trì kết nối này vĩnh viễn mở và sẽ sử dụng nó để gửi các bản cập nhật dưới dạng các cuộc gọi đến các chức năng kịch bản được xác định tại client. Theo cách nào đó, chúng ta có thể nói rằng kỹ thuật này bao gồm các kịch bản lệnh trực tuyến được thực hiện tại client khi chúng nhận được.

Bởi vì kết nối được giữ mở vĩnh viễn, các nguồn lực được sử dụng hiệu quả hơn bởi vì chúng không bị lãng phí trong quá trình kết nối và ngắt kết nối. Vì vậy, thực tế có thể đạt được thời gian thực của coveted trong hướng server-client.

Giống như trong kỹ thuật trước đó, việc sử dụng HTML, JavaScript và HTTP làm cho phạm vi ứng dụng của nó hầu như phổ biến, mặc dù rõ ràng là rất hướng về phía client hỗ trợ những công nghệ đó, chẳng hạn như các trình duyệt web. Nghĩa là, việc thực hiện các loại client khác, chẳng hạn như ứng dụng máy tính để bàn, hoặc các quá trình khác hoạt động như người tiêu dùng của các dịch vụ đó sẽ khá phức tạp, như thể hiện trong hình 2-7.



Kỹ thuật này cũng không được miễn trừ những bất lợi. Trong quá trình thực hiện, cần phải lưu ý rằng có thể xảy ra lỗi thời gian do client, server hoặc yếu tố trung gian (như proxy và tường lửa) gây ra. Ngoài ra, để có được trải nghiệm thời gian thực tốt nhất, phản hồi phải được gửi đến client ngay lập tức và không được giữ lại trong bộ đệm hoặc bộ nhớ cache. Và, bởi vì các phản hồi sẽ tích lũy bên trong khung nội tuyến, trong bộ nhớ của client, chúng ta có thể sẽ chiếm quá nhiều RAM, vì vậy chúng ta phải "tái chế" hoặc loại bỏ nội dung định kỳ.

Cuối cùng, thực tế là kết nối được sử dụng chỉ để gửi dữ liệu từ server tới client làm cho nó cần thiết phải sử dụng một kết nối bổ sung khi chúng ta muốn gửi nó theo hướng ngược lại - nghĩa là từ client đến server.

**The world needs more than just push**

Cho đến nay, chúng ta đã thấy những kỹ thuật cho phép chúng ta đạt được sự thúc đẩy, nghĩa là chúng cho phép server có thể gửi thông tin đến client không đồng bộ khi nó được tạo ra. Đã có sáng kiến ​​cho một yếu tố thông thường sẽ đảm nhận vai trò thụ động trong giao tiếp với client.

Tuy nhiên, trong bối cảnh các ứng dụng không đồng bộ, đa người dùng và thời gian thực, đẩy là một trong những khía cạnh không thể thiếu. Để tạo ra những hệ thống này luôn luôn đáng ngạc nhiên, chúng ta cần nhiều khả năng hơn. Ở đây chúng tôi liệt kê một vài trong điểm:

* Quản lý người dùng kết nối server phải luôn biết người dùng nào được kết nối với các dịch vụ, những người đó ngắt kết nối và về cơ bản phải kiểm soát tất cả các khía cạnh liên quan đến giám sát một số lượng client không xác định.
* Quản lý đăng ký server phải có khả năng quản lý "đăng ký" hoặc nhóm các client muốn nhận các loại tin nhắn cụ thể. Ví dụ: trong dịch vụ phòng trò chuyện, chỉ những người dùng kết nối với một phòng cụ thể mới nhận được thông báo được gửi đến phòng đó. Bằng cách này, việc phân phối thông tin được tối ưu hóa và client không nhận được thông tin không liên quan đến chúng, giảm thiểu lãng phí nguồn lực.
* Nhận và xử lý các hành động các server có khả năng không chỉ gửi thông tin cho client trong thời gian thực mà còn nhận nó và xử lý nó trên bay.
* Giám sát các bài nộp Bởi vì không thể bảo đảm rằng tất cả các client kết nối trong cùng một điều kiện, có thể có các kết nối ở các tốc độ khác nhau, sự không ổn định đường dây, hoặc sự cố đột xuất, và điều này có nghĩa là nó cần phải cung cấp cho các cơ chế có khả năng xếp hàng và quản lý thông tin Riêng biệt để đảm bảo rằng tất cả client đều được cập nhật.
* Cung cấp một API linh hoạt, có khả năng tiêu thụ dễ dàng bởi nhiều client, điều này thậm chí còn đúng hơn ngày nay, khi có rất nhiều thiết bị từ đó có thể truy cập các dịch vụ trực tuyến.

Chắc chắn có thể liệt kê nhiều hơn, nhưng những ví dụ này là quá đủ để cho bạn một ý tưởng về sự phức tạp vốn có trong việc phát triển các loại ứng dụng.