O'REILLY®

Software Architecture Patterns

Mark Richards

O'REILLY®

Thêm vào Tài nguyên

4 cách dễ dàng để tìm hiểu thêm và cập nhật

Bản tin lập trình Nhận tin tức
và nội dung liên quan đến lập trình được gửi hàng tuần tới hộp thư đến
của bạn. oreilly.com/programming/newsletter

Chuỗi webcast miễn phí Tìm
hiểu về các chủ đề lập trình phổ biến từ các chuyên gia trực tiếp, trực
tuyến. webcasts.oreilly.com

O'Reilly Radar
Đọc thêm thông tin chi tiết và phân tích về các công nghệ mới
nổi. radar.oreilly.com

Hội nghị Đắm
mình trong việc học tại hội nghị O'Reilly sắp tới. hội
nghị.oreilly.com

Kiến trúc phần mềm mẫu

Hiểu kiến trúc chung
Các mẫu và thời điểm sử dụng Tem

Mark Richards

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo O'REILLY®

Các mẫu kiến trúc phần mềm của Mark

Richards

Được in tại Hoa Kỳ.

Được xuất bản bởi O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Biên tập: Heather Scherer

Biên tập sản xuất: Colleen Lobner

Biên tập viên: Amanda Kersey

Nhà thiết kế nội thất: David Futato

Thiết kế bìa: Ellie Volckhausen

Người minh họa: Rebecca Demarest

Tháng 2 năm 2015: Ấn bản đầu tiên

Lịch sử sửa đổi cho lần xuất bản đầu tiên

24-02-2015: Phát hành lần đầu

30-03-2015: Phát hành lần thứ hai

978-1-491-92424-2

[LSI]

Mục lục

Giới thiệu

Việc các nhà phát triển bắt đầu viết mã một ứng dụng mà không có kiến trúc chính thức là điều quá phổ biến. Nếu không có kiến trúc rõ ràng và được xác định rõ ràng, hầu hết các nhà phát triển và kiến trúc sư sẽ sử dụng mẫu kiến trúc phân lớp truyền thống tiêu chuẩn trên thực tế (còn gọi là kiến trúc n tầng), tạo ra các lớp ẩn bằng cách tách các mô-đun mã nguồn thành các gói. Thật không may, kết quả thường thấy của cách làm này là một tập hợp các mô-đun mã nguồn không có tổ chức, thiếu vai trò, trách nhiệm và mối quan hệ rõ ràng với nhau. Điều này thường được gọi là quả bóng lớn của kiến trúc bùn phản mô hình.

Các ứng dụng thiếu kiến trúc hình thức thường có tính liên kết chặt chẽ, dễ vỡ, khó thay đổi và không có tầm nhìn hoặc phương hướng rõ ràng. Kết quả là rất khó xác định các đặc điểm kiến trúc của ứng dụng nếu không hiểu đầy đủ hoạt động bên trong của mọi thành phần và mô-đun trong hệ thống.
Các câu hỏi cơ bản về triển khai và bảo trì rất khó trả lời: Kiến trúc có quy mô không? Các đặc tính hiệu suất của ứng dụng là gì? Ứng dụng phản ứng với sự thay đổi dễ dàng như thế nào? Các đặc điểm triển khai của ứng dụng là gì? Kiến trúc đáp ứng như thế nào?

Các mẫu kiến trúc giúp xác định các đặc điểm và hành vi cơ bản của một ứng dụng. Ví dụ, một số mẫu kiến trúc tự nhiên phù hợp với các ứng dụng có khả năng mở rộng cao, trong khi các mẫu kiến trúc khác lại phù hợp với các ứng dụng có tính linh hoạt cao một cách tự nhiên. Biết được đặc điểm, điểm mạnh và điểm yếu của từng mẫu kiến trúc là cần thiết.

sary để chọn một trong những phù hợp với doanh nghiệp cụ thể của bạn
nhu cầu và mục tiêu.

Là một kiến trúc sư, bạn phải luôn biện minh cho các quyết định kiến trúc của mình,
đặc biệt khi nói đến việc lựa chọn một kiểu kiến trúc cụ thể
chim nhạn hoặc cách tiếp cận. Mục tiêu của báo cáo này là cung cấp cho bạn đủ thông tin
cơ sở để đưa ra và biện minh cho quyết định đó.

CHƯƠNG 1

Kiến trúc lớp

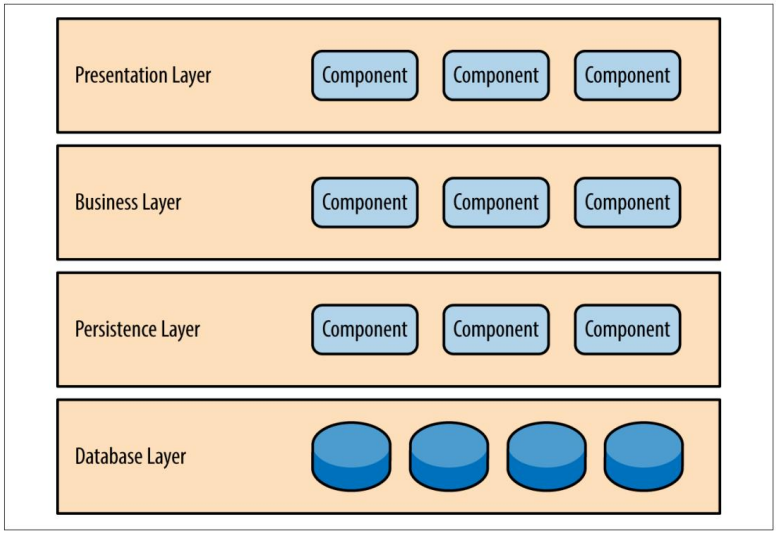
Mẫu kiến trúc phổ biến nhất là mẫu kiến trúc phân lớp, hay còn gọi là mẫu kiến trúc n tầng. Mẫu này là tiêu chuẩn thực tế cho hầu hết các ứng dụng Java EE và do đó được hầu hết các kiến trúc sư, nhà thiết kế và nhà phát triển biết đến rộng rãi. Mẫu kiến trúc phân lớp phù hợp chặt chẽ với cấu trúc tổ chức và truyền thông CNTT truyền thống được tìm thấy ở hầu hết các công ty, khiến nó trở thành một lựa chọn tự nhiên cho hầu hết các nỗ lực phát triển ứng dụng kinh doanh.

Mô tả mẫu

Các thành phần trong mẫu kiến trúc phân lớp được tổ chức thành các lớp ngang, mỗi lớp thực hiện một vai trò cụ thể trong ứng dụng (ví dụ: logic trình bày hoặc logic nghiệp vụ). Mặc dù mẫu kiến trúc phân lớp không chỉ định số lượng và loại lớp phải tồn tại trong mẫu, nhưng hầu hết các kiến trúc phân lớp đều bao gồm bốn lớp tiêu chuẩn: trình bày, nghiệp vụ, kiên trì và cơ sở dữ liệu (Hình 1-1). Trong một số trường hợp, lớp nghiệp vụ và lớp kiên trì được kết hợp thành một lớp nghiệp vụ duy nhất, đặc biệt khi logic bền vững (ví dụ: SQL hoặc HSQL) được nhúng trong các thành phần của lớp nghiệp vụ. Do đó, các ứng dụng nhỏ hơn có thể chỉ có ba lớp, trong khi các ứng dụng kinh doanh lớn hơn và phức tạp hơn có thể chứa năm lớp trở lên.

Mỗi lớp của mẫu kiến trúc phân lớp có một vai trò và trách nhiệm cụ thể trong ứng dụng. Ví dụ: lớp trình bày sẽ chịu trách nhiệm xử lý tất cả giao diện người dùng và

logic giao tiếp của trình duyệt, trong khi lớp nghiệp vụ sẽ chịu trách nhiệm thực thi các quy tắc nghiệp vụ cụ thể được liên kết với yêu cầu. Mỗi lớp trong kiến trúc tạo thành một sự trừu tượng xung quanh công việc cần được thực hiện để đáp ứng một yêu cầu kinh doanh cụ thể. Ví dụ: lớp trình bày không cần biết hoặc lo lắng về cách lấy dữ liệu khách hàng; nó chỉ cần hiển thị thông tin đó trên màn hình ở định dạng cụ thể. Tương tự, lớp nghiệp vụ không cần quan tâm đến cách định dạng dữ liệu khách hàng để hiển thị trên màn hình hoặc thậm chí dữ liệu khách hàng đến từ đâu; nó chỉ cần lấy dữ liệu từ lớp kiên trì, thực hiện logic nghiệp vụ dựa trên dữ liệu (ví dụ: tính toán các giá trị hoặc dữ liệu tổng hợp) và chuyển thông tin đó lên lớp trình bày.

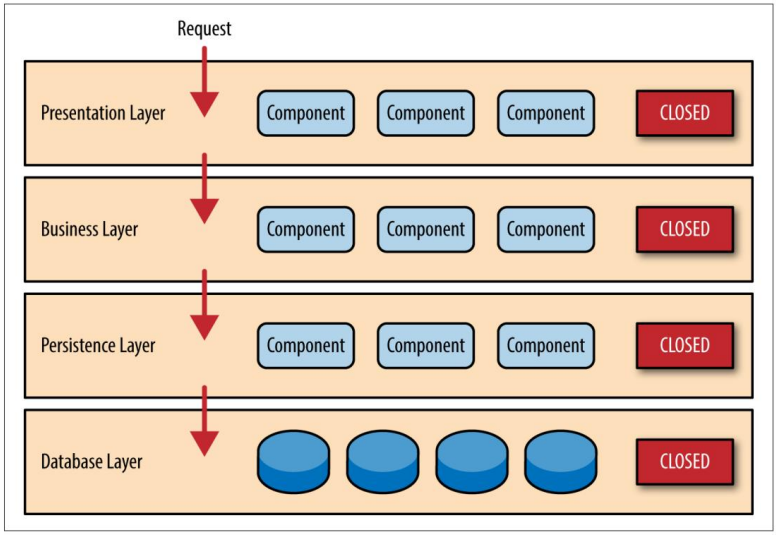


Hình 1-1. Mô hình kiến trúc phân lớp

Một trong những tính năng mạnh mẽ của mẫu kiến trúc phân lớp là sự phân tách mối quan tâm giữa các thành phần. Các thành phần trong một lớp cụ thể chỉ xử lý logic liên quan đến lớp đó. Ví dụ, các thành phần trong lớp trình bày chỉ xử lý logic trình bày, trong khi các thành phần nằm trong lớp nghiệp vụ chỉ xử lý logic nghiệp vụ. Kiểu phân loại thành phần này giúp bạn dễ dàng xây dựng các mô hình vai trò và trách nhiệm hiệu quả trong kiến trúc của mình, đồng thời cũng giúp bạn dễ dàng phát triển, kiểm tra, quản lý và bảo trì các ứng dụng sử dụng mẫu kiến trúc này do giao diện thành phần được xác định rõ ràng và phạm vi thành phần hạn chế.

Các khái niệm chính

Lưu ý trong Hình 1-2 rằng mỗi lớp trong kiến trúc được đánh dấu là bị đóng. Đây là một khái niệm rất quan trọng trong mẫu kiến trúc phân lớp. Lớp đóng có nghĩa là khi một yêu cầu di chuyển từ lớp này sang lớp khác, nó phải đi qua lớp ngay bên dưới nó để đến lớp tiếp theo bên dưới lớp đó. Ví dụ: một yêu cầu bắt nguồn từ lớp trình bày trước tiên phải đi qua lớp nghiệp vụ và sau đó đến lớp kiên trì trước khi đến lớp cơ sở dữ liệu.



Hình 1-2. Các lớp đã đóng và yêu cầu quyền truy cập

Vậy tại sao không cho phép lớp trình bày truy cập trực tiếp vào lớp lưu trữ hoặc lớp cơ sở dữ liệu? Xét cho cùng, việc truy cập cơ sở dữ liệu trực tiếp từ lớp trình bày nhanh hơn nhiều so với việc đi qua một loạt các lớp không cần thiết chỉ để truy xuất hoặc lưu thông tin cơ sở dữ liệu. Câu trả lời cho câu hỏi này nằm ở một khái niệm quan trọng được gọi là các lớp cách ly.

Khái niệm lớp cách ly có nghĩa là những thay đổi được thực hiện trong một lớp của kiến trúc thường không tác động hoặc ảnh hưởng đến các thành phần trong các lớp khác: thay đổi được cách ly đối với các thành phần trong lớp đó và có thể là một lớp liên kết khác (chẳng hạn như lớp kiên trì chứa SQL). Nếu bạn cho phép lớp trình bày truy cập trực tiếp vào lớp lưu trữ lâu dài thì các thay đổi được thực hiện đối với SQL trong

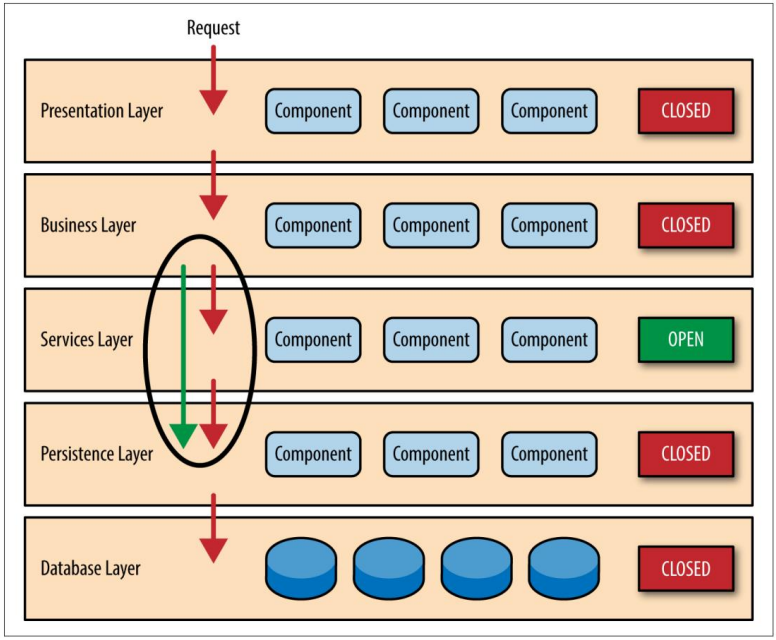
lớp kiên trì sẽ tác động đến cả lớp nghiệp vụ và lớp trình bày, do đó tạo ra một ứng dụng được liên kết rất chặt chẽ với nhiều sự phụ thuộc lẫn nhau giữa các thành phần. Kiểu kiến trúc này sau đó trở nên rất khó thay đổi và tốn kém.

Khái niệm lớp cách ly cũng có nghĩa là mỗi lớp độc lập với các lớp khác, do đó có rất ít hoặc không có kiến thức về hoạt động bên trong của các lớp khác trong kiến trúc. Để hiểu sức mạnh và tầm quan trọng của khái niệm này, hãy xem xét nỗ lực tái cấu trúc lớn để chuyển đổi khung trình bày từ JSP (Trang máy chủ Java) sang JSF (Các mặt máy chủ Java). Giả sử rằng các hợp đồng (ví dụ: mô hình) được sử dụng giữa lớp trình bày và lớp nghiệp vụ vẫn giữ nguyên, lớp nghiệp vụ không bị ảnh hưởng bởi việc tái cấu trúc và vẫn hoàn toàn độc lập với loại khung giao diện người dùng được lớp trình bày sử dụng .

Mặc dù các lớp đóng tạo điều kiện thuận lợi cho các lớp cách ly và do đó giúp cách ly sự thay đổi trong kiến trúc, nhưng đôi khi việc mở một số lớp nhất định là điều hợp lý. Ví dụ: giả sử bạn muốn thêm một lớp dịch vụ chia sẻ vào một kiến trúc chứa các thành phần dịch vụ chung được truy cập bởi các thành phần trong lớp nghiệp vụ (ví dụ: các lớp tiện ích dữ liệu và chuỗi hoặc các lớp kiểm tra và ghi nhật ký). Tạo một lớp dịch vụ thường là một ý tưởng hay trong trường hợp này vì về mặt kiến trúc, nó hạn chế quyền truy cập vào các dịch vụ được chia sẻ đối với lớp nghiệp vụ (chứ không phải lớp trình bày). Nếu không có lớp riêng biệt thì về mặt kiến trúc sẽ không có gì hạn chế lớp trình bày truy cập vào các dịch vụ phổ biến này, gây khó khăn cho việc quản lý hạn chế truy cập này.

Trong ví dụ này, lớp dịch vụ mới có thể nằm bên dưới lớp nghiệp vụ để chỉ ra rằng các thành phần trong lớp dịch vụ này không thể truy cập được từ lớp trình bày. Tuy nhiên, điều này gây ra một vấn đề ở chỗ lớp nghiệp vụ hiện bắt buộc phải đi qua lớp dịch vụ để đến lớp lưu trữ, điều này không có ý nghĩa gì cả. Đây là một vấn đề lâu đời với kiến trúc phân lớp và được giải quyết bằng cách tạo các lớp mở bên trong kiến trúc.

Như minh họa trong Hình 1-3, lớp dịch vụ trong trường hợp này được đánh dấu là mở, nghĩa là các yêu cầu được phép bỏ qua lớp mở này và đi thẳng đến lớp bên dưới nó. Trong ví dụ sau, vì lớp dịch vụ được mở nên lớp nghiệp vụ hiện được phép bỏ qua nó và đi thẳng đến lớp kiên trì, điều này hoàn toàn hợp lý.



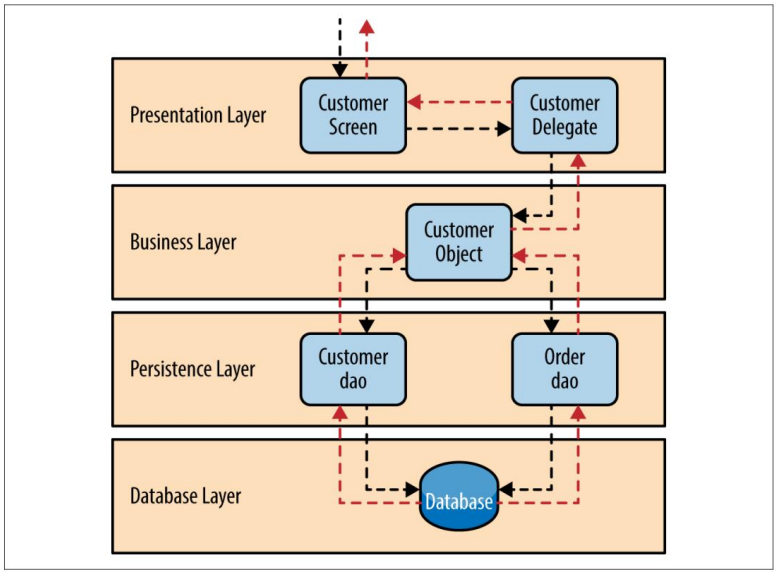
Hình 1-3. Mở lớp và yêu cầu fow

Việc tận dụng khái niệm lớp mở và lớp đóng giúp xác định mối quan hệ giữa các lớp kiến trúc và luồng yêu cầu, đồng thời cũng cung cấp cho các nhà thiết kế và nhà phát triển thông tin cần thiết để hiểu các hạn chế truy cập lớp khác nhau trong kiến trúc. Việc không ghi lại hoặc truyền đạt đúng cách các lớp nào trong kiến trúc là mở và đóng (và tại sao) thường dẫn đến các kiến trúc được liên kết chặt chẽ và dễ vỡ, rất khó kiểm tra, bảo trì và triển khai.

Ví dụ về mẫu

Để minh họa cách hoạt động của kiến trúc phân lớp, hãy xem xét yêu cầu từ người dùng doanh nghiệp để lấy thông tin khách hàng cho một cá nhân cụ thể như minh họa trong Hình 1-4. Mũi tên màu đen hiển thị yêu cầu chuyển xuống cơ sở dữ liệu để truy xuất dữ liệu khách hàng và mũi tên màu đỏ hiển thị phản hồi quay ngược lên màn hình để hiển thị dữ liệu. Trong ví dụ này, thông tin khách hàng bao gồm cả dữ liệu khách hàng và dữ liệu đơn hàng (đơn hàng do khách hàng đặt).

Màn hình khách hàng có nhiệm vụ tiếp nhận yêu cầu và hiển thị thông tin khách hàng. Nó không biết dữ liệu ở đâu, được lấy ra như thế nào hoặc phải truy vấn bao nhiêu bảng cơ sở dữ liệu để lấy dữ liệu. Khi màn hình khách hàng nhận được yêu cầu lấy thông tin khách hàng cho một cá nhân cụ thể, nó sẽ chuyển tiếp yêu cầu đó đến mô-đun đại biểu khách hàng. Mô-đun này chịu trách nhiệm biết mô-đun nào trong lớp nghiệp vụ có thể xử lý yêu cầu đó cũng như cách truy cập mô-đun đó và dữ liệu nào nó cần (hợp đồng). Đối tượng khách hàng trong lớp nghiệp vụ có trách nhiệm tổng hợp tất cả thông tin cần thiết theo yêu cầu nghiệp vụ (trong trường hợp này là để lấy thông tin khách hàng). Mô-đun này gọi mô-đun dao khách hàng (đối tượng truy cập dữ liệu) trong lớp lưu giữ để lấy dữ liệu khách hàng và mô-đun dao đặt hàng để lấy thông tin đơn hàng. Các mô-đun này lần lượt thực thi các câu lệnh SQL để lấy dữ liệu tương ứng và chuyển nó trở lại đối tượng khách hàng trong lớp nghiệp vụ. Khi đối tượng khách hàng nhận được dữ liệu, nó sẽ tổng hợp dữ liệu và chuyển thông tin đó trở lại cho đại biểu khách hàng, sau đó đại diện khách hàng sẽ chuyển dữ liệu đó đến màn hình khách hàng để hiển thị cho người dùng.



Hình 1-4. Ví dụ về kiến trúc phân lớp

Từ góc độ công nghệ, có hàng tá cách để triển khai các mô-đun này. Ví dụ: trong nền tảng Java, màn hình khách hàng có thể là màn hình Java Server Faces (JSF)

được kết hợp với đại biểu khách hàng làm thành phần Bean được quản lý. Đối tượng khách hàng trong lớp nghiệp vụ có thể là một Spring Bean cục bộ hoặc một Bean EJB3 từ xa. Các đối tượng truy cập dữ liệu được minh họa trong ví dụ trước có thể được triển khai dưới dạng POJO đơn giản (Đối tượng Java thuần túy), tệp MyBatis XML Mapper hoặc thậm chí các đối tượng đóng gói lệnh gọi JDBC thô hoặc truy vấn Hibernate. Từ góc độ nền tảng Micro-soft, màn hình khách hàng có thể là một mô-đun ASP (các trang máy chủ đang hoạt động) sử dụng .NET framework để truy cập các mô-đun C# trong lớp nghiệp vụ, với các mô-đun truy cập dữ liệu của khách hàng và đơn đặt hàng được triển khai dưới dạng ADO (ActiveX Đối tượng dữ liệu).

Cân nhắc

Mẫu kiến trúc phân lớp là một mẫu có mục đích chung vững chắc, khiến nó trở thành điểm khởi đầu tốt cho hầu hết các ứng dụng, đặc biệt khi bạn không chắc chắn mẫu kiến trúc nào phù hợp nhất cho ứng dụng của mình. Tuy nhiên, có một số điều cần xem xét từ quan điểm kiến trúc khi chọn mẫu này.

Điều đầu tiên cần chú ý là cái được gọi là mô hình chống hố sụt kiến trúc. Chống mẫu này mô tả tình huống trong đó các yêu cầu chảy qua nhiều lớp của kiến trúc dưới dạng xử lý chuyển tiếp đơn giản với rất ít hoặc không có logic nào được thực hiện trong mỗi lớp. Ví dụ: giả sử lớp trình bày đáp ứng yêu cầu từ người dùng để truy xuất dữ liệu khách hàng. Lớp trình bày chuyển yêu cầu đến lớp nghiệp vụ, lớp này chỉ chuyển yêu cầu đến lớp lưu giữ lâu dài, sau đó thực hiện lệnh gọi SQL đơn giản đến lớp cơ sở dữ liệu để truy xuất dữ liệu khách hàng. Sau đó, dữ liệu sẽ được chuyển trở lại ngăn xếp mà không cần xử lý hoặc logic bổ sung để tổng hợp, tính toán hoặc chuyển đổi dữ liệu.

Mỗi kiến trúc phân lớp sẽ có ít nhất một số kịch bản rơi vào mô hình chống hố sụt kiến trúc. Tuy nhiên, điều quan trọng là phân tích tỷ lệ phần trăm yêu cầu thuộc danh mục này. Quy tắc 80-20 thường là một phương pháp hay nên tuân theo để xác định xem bạn có đang gặp phải mô hình chống hố sụt kiến trúc hay không. Thông thường có khoảng 20% yêu cầu được xử lý chuyển tiếp đơn giản và 80% yêu cầu có một số logic nghiệp vụ liên quan đến yêu cầu. Tuy nhiên, nếu bạn nhận thấy tỷ lệ này bị đảo ngược và phần lớn các yêu cầu của bạn chỉ được xử lý chuyển tiếp đơn giản, bạn có thể muốn xem xét thực hiện một số

Các lớp kiến trúc mở, hãy nhớ rằng sẽ khó kiểm soát sự thay đổi hơn do thiếu sự cô lập lớp.

Một điều cần cân nhắc khác với mẫu kiến trúc phân lớp là nó có xu hướng phù hợp với các ứng dụng nguyên khối, ngay cả khi bạn chia lớp trình bày và lớp nghiệp vụ thành các đơn vị có thể triển khai riêng biệt. Mặc dù điều này có thể không phải là mối lo ngại đối với một số ứng dụng nhưng nó đặt ra một số vấn đề tiềm ẩn về mặt triển khai, độ bền và độ tin cậy nói chung, hiệu suất và khả năng mở rộng.

Phân tích mẫu

Bảng sau đây chứa xếp hạng và phân tích các đặc điểm kiến trúc chung cho mẫu kiến trúc phân lớp. Xếp hạng cho từng đặc điểm dựa trên xu hướng tự nhiên đối với đặc điểm đó như một khả năng dựa trên việc triển khai mẫu điển hình, cũng như mục đích chung của mẫu. Để so sánh song song mối liên hệ của mẫu này với các mẫu khác trong báo cáo này, vui lòng tham khảo Phụ lục A ở cuối báo cáo này.

Đánh giá về tính linh hoạt tổng thể : Phân tích thấp: Tính linh hoạt tổng thể là khả năng phản ứng nhanh chóng với môi trường thay đổi liên tục. Mặc dù sự thay đổi có thể được tách biệt thông qua các lớp tính năng cách ly của mẫu này, nhưng việc thực hiện các thay đổi trong mẫu kiến trúc này vẫn cồng kềnh và tốn thời gian do tính chất nguyên khối của hầu hết các triển khai cũng như sự kết hợp chặt chẽ của các thành phần thường thấy với mẫu này. mẫu.

Dễ triển khai Xếp hạng : Phân tích thấp: Tùy thuộc vào cách bạn triển khai mẫu này, việc triển khai có thể trở thành một vấn đề, đặc biệt đối với các ứng dụng lớn hơn. Một thay đổi nhỏ đối với một thành phần có thể yêu cầu triển khai lại toàn bộ ứng dụng (hoặc một phần lớn của ứng dụng), dẫn đến việc triển khai cần được lên kế hoạch, lên lịch và thực hiện ngoài giờ làm việc hoặc vào cuối tuần.
Do đó, mô hình này không dễ dàng phù hợp với quy trình phân phối liên tục, điều này càng làm giảm xếp hạng tổng thể cho việc triển khai.

Xếp hạng
khả năng
kiểm tra : Phân tích cao: Vì các thành phần thuộc về các lớp cụ thể trong kiến trúc nên các lớp khác có thể bị mô phỏng hoặc sơ khai, khiến cho mẫu này tương đối dễ kiểm tra. Nhà phát triển có thể mô phỏng thành phần hoặc màn hình trình bày để tách thử nghiệm trong thành phần nghiệp vụ, cũng như mô phỏng lớp nghiệp vụ để kiểm tra chức năng màn hình nhất định.

Xếp hạng hiệu
suất : Phân
tích thấp: Mặc dù đúng là một số kiến trúc phân lớp có thể hoạt động tốt, nhưng mẫu này không phù hợp với các ứng dụng hiệu suất cao do sự thiếu hiệu quả khi phải trải qua nhiều lớp kiến trúc để thực hiện yêu cầu kinh doanh.

Xếp hạng
khả năng mở
rộng : Phân tích thấp: Do xu hướng triển khai nguyên khối và liên kết chặt chẽ của mẫu này, nên việc xây dựng các ứng dụng sử dụng mẫu kiến trúc này thường khó mở rộng quy mô. Bạn có thể mở rộng kiến trúc phân lớp bằng cách chia các lớp thành các triển khai vật lý riêng biệt hoặc sao chép toàn bộ ứng dụng thành nhiều nút, nhưng nhìn chung mức độ chi tiết quá rộng, khiến việc mở rộng quy mô trở nên tốn kém.

Đánh giá về mức độ
dễ phát
triển : Phân tích cao: Mức độ dễ phát triển đạt điểm tương đối cao, chủ yếu là do mẫu này quá nổi tiếng và không quá phức tạp để thực hiện. Bởi vì hầu hết các công ty phát triển ứng dụng bằng cách tách các bộ kỹ năng theo lớp (trình bày, kinh doanh, cơ sở dữ liệu), nên mô hình này trở thành một lựa chọn tự nhiên cho hầu hết việc phát triển ứng dụng kinh doanh. Mối liên hệ giữa cơ cấu tổ chức và truyền thông của công ty với cách công ty phát triển phần mềm được phác thảo ra cái được gọi là định luật Conway. Bạn có thể Google “định luật Conway” để biết thêm thông tin về mối tương quan hấp dẫn này.

CHƯƠNG 2

Kiến trúc hướng sự kiện

Mẫu kiến trúc hướng sự kiện là mẫu kiến trúc không đồng bộ phân tán phổ biến được sử dụng để tạo ra các ứng dụng có khả năng mở rộng cao. Nó cũng có khả năng thích ứng cao và có thể được sử dụng cho các ứng dụng nhỏ cũng như các ứng dụng lớn, phức tạp. Kiến trúc hướng sự kiện được tạo thành từ các thành phần xử lý sự kiện có mục đích đơn, tách rời cao, nhận và xử lý không đồng bộ sự kiện.

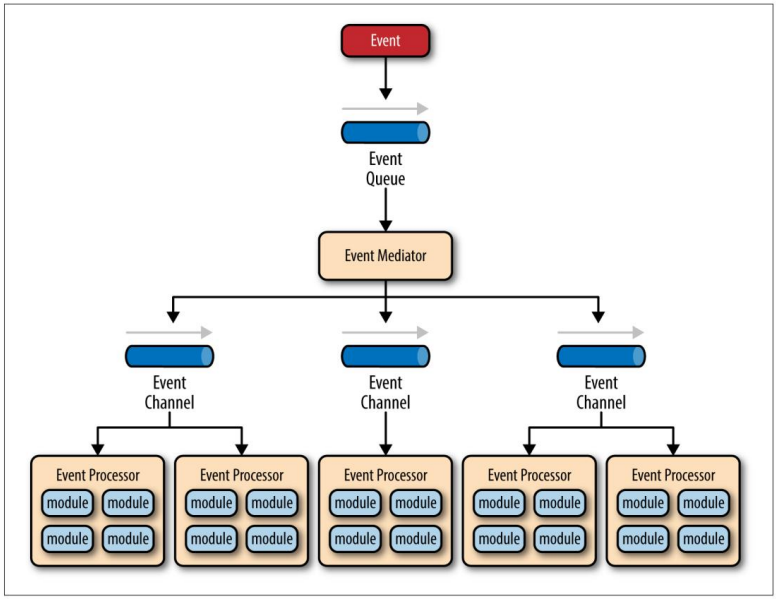
Mẫu kiến trúc hướng sự kiện bao gồm hai cấu trúc liên kết chính, bộ hòa giải và bộ môi giới. Cấu trúc liên kết của người hòa giải thường được sử dụng khi bạn cần sắp xếp nhiều bước trong một sự kiện thông qua một người hòa giải trung tâm, trong khi cấu trúc liên kết của người môi giới được sử dụng khi bạn muốn xâu chuỗi các sự kiện lại với nhau mà không cần sử dụng người hòa giải trung tâm. Bởi vì các đặc điểm kiến trúc và chiến lược triển khai khác nhau giữa hai cấu trúc liên kết này, điều quan trọng là phải hiểu từng cấu trúc để biết cái nào phù hợp nhất cho tình huống cụ thể của bạn.

Cấu trúc liên kết hòa giải

Cấu trúc liên kết hòa giải rất hữu ích cho các sự kiện có nhiều bước và yêu cầu một số mức độ điều phối để xử lý sự kiện. Ví dụ: một sự kiện duy nhất để thực hiện giao dịch chứng khoán có thể yêu cầu bạn xác thực giao dịch trước tiên, sau đó kiểm tra sự tuân thủ của giao dịch chứng khoán đó với các quy tắc tuân thủ khác nhau, chỉ định giao dịch cho nhà môi giới, tính toán hoa hồng và cuối cùng đặt giao dịch. giao dịch với nhà môi giới đó. Tất cả các bước này sẽ yêu cầu một mức độ phối hợp nào đó để ngăn chặn

khai thác thứ tự các bước và những bước nào có thể được thực hiện tuần tự và song song.

Có bốn loại thành phần kiến trúc chính trong cấu trúc liên kết của bộ hòa giải: hàng đợi sự kiện, bộ hòa giải sự kiện, kênh sự kiện và bộ xử lý sự kiện. Luồng sự kiện bắt đầu bằng việc máy khách gửi sự kiện đến hàng đợi sự kiện, hàng đợi này được sử dụng để vận chuyển sự kiện đến người hòa giải sự kiện. Người hòa giải sự kiện nhận sự kiện ban đầu và điều phối sự kiện đó bằng cách gửi các sự kiện không đồng bộ bổ sung đến các kênh sự kiện để thực thi từng bước của quy trình. Bộ xử lý sự kiện lắng nghe trên các kênh sự kiện, nhận sự kiện từ bộ hòa giải sự kiện và thực thi logic nghiệp vụ cụ thể để xử lý sự kiện. Hình 2-1 minh họa cấu trúc liên kết trung gian chung của mẫu kiến trúc hướng sự kiện.



Hình 2-1. Cấu trúc liên kết trung gian kiến trúc hướng sự kiện

Thông thường có từ hàng chục đến hàng trăm hàng đợi sự kiện trong kiến trúc hướng sự kiện. Mẫu không chỉ định việc triển khai thành phần hàng đợi sự kiện; nó có thể là hàng đợi tin nhắn, điểm cuối dịch vụ web hoặc bất kỳ sự kết hợp nào của chúng.

Có hai loại sự kiện trong mẫu này: sự kiện ban đầu và sự kiện xử lý. Sự kiện ban đầu là sự kiện ban đầu được nhận bởi

người hòa giải, trong khi các sự kiện xử lý là những sự kiện được tạo ra bởi người hòa giải và được các thành phần xử lý sự kiện nhận.

Thành phần trung gian sự kiện chịu trách nhiệm điều phối các bước có trong sự kiện ban đầu. Đối với mỗi bước trong sự kiện ban đầu, bộ hòa giải sự kiện sẽ gửi một sự kiện xử lý cụ thể đến một kênh sự kiện, sau đó được bộ xử lý sự kiện nhận và xử lý. Điều quan trọng cần lưu ý là trình hòa giải sự kiện không thực sự thực hiện logic nghiệp vụ cần thiết để xử lý sự kiện ban đầu; đúng hơn, nó biết các bước cần thiết để xử lý sự kiện ban đầu.

Các kênh sự kiện được người hòa giải sự kiện sử dụng để truyền không đồng bộ các sự kiện xử lý cụ thể liên quan đến từng bước trong sự kiện ban đầu tới bộ xử lý sự kiện. Các kênh sự kiện có thể là hàng đợi tin nhắn hoặc chủ đề tin nhắn, mặc dù các chủ đề tin nhắn được sử dụng rộng rãi nhất với cấu trúc liên kết hòa giải để các sự kiện xử lý có thể được xử lý bởi nhiều bộ xử lý sự kiện (mỗi bộ xử lý thực hiện một nhiệm vụ khác nhau dựa trên sự kiện xử lý nhận được).

Các thành phần bộ xử lý sự kiện chứa logic nghiệp vụ ứng dụng cần thiết để xử lý sự kiện xử lý. Bộ xử lý sự kiện là các thành phần kiến trúc độc lập, độc lập, có tính tách rời cao, thực hiện một tác vụ cụ thể trong ứng dụng hoặc hệ thống.
Mặc dù mức độ chi tiết của thành phần bộ xử lý sự kiện có thể thay đổi từ chi tiết (ví dụ: tính thuế bán hàng cho một đơn hàng) đến chi tiết thô (ví dụ: xử lý yêu cầu bảo hiểm), điều quan trọng cần lưu ý là nhìn chung, mỗi Thành phần bộ xử lý sự kiện phải thực hiện một nhiệm vụ kinh doanh duy nhất và không dựa vào các bộ xử lý sự kiện khác để hoàn thành nhiệm vụ cụ thể của nó.

Trình hòa giải sự kiện có thể được thực hiện theo nhiều cách khác nhau. Với tư cách là kiến trúc sư, bạn nên hiểu từng tùy chọn triển khai này để đảm bảo rằng giải pháp bạn chọn cho người hòa giải sự kiện phù hợp với nhu cầu và yêu cầu của bạn.

Việc triển khai đơn giản và phổ biến nhất của trình hòa giải sự kiện là thông qua các trung tâm tích hợp nguồn mở như Spring Integration, Apache Camel hoặc Mule ESB. Các luồng sự kiện trong các trung tâm tích hợp nguồn mở này thường được triển khai thông qua mã Java hoặc DSL (ngôn ngữ dành riêng cho miền). Để dàn xếp và điều phối phức tạp hơn, bạn có thể sử dụng BPEL (ngôn ngữ thực thi quy trình nghiệp vụ) kết hợp với công cụ BPEL chẳng hạn như nguồn mở

Apache ODE. BPEL là ngôn ngữ giống XML tiêu chuẩn mô tả dữ liệu và các bước cần thiết để xử lý một sự kiện ban đầu. Đối với các ứng dụng rất lớn yêu cầu điều phối phức tạp hơn nhiều (bao gồm các bước liên quan đến tương tác của con người), bạn có thể triển khai trình hòa giải sự kiện bằng cách sử dụng trình quản lý quy trình nghiệp vụ (BPM), chẳng hạn như jBPM.

Hiểu nhu cầu của bạn và kết hợp chúng với việc triển khai trình hòa giải sự kiện chính xác là rất quan trọng đối với sự thành công của bất kỳ kiến trúc hướng sự kiện nào sử dụng cấu trúc liên kết này. Việc sử dụng một trung tâm tích hợp nguồn mở để thực hiện việc điều phối quản lý quy trình kinh doanh rất phức tạp là công thức dẫn đến thất bại, giống như việc triển khai giải pháp BPM để thực hiện logic định tuyến đơn giản.

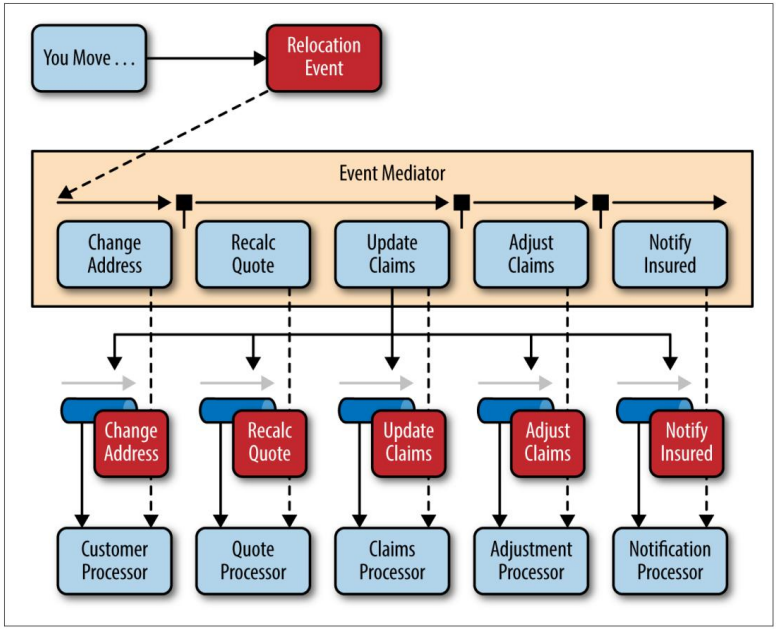
Để minh họa cách hoạt động của cấu trúc liên kết hòa giải, giả sử bạn được bảo hiểm thông qua một công ty bảo hiểm và bạn quyết định chuyển đi. Trong trường hợp này, sự kiện ban đầu có thể được gọi là sự kiện di dời. Các bước liên quan đến việc xử lý một sự kiện di dời được chứa trong bộ hòa giải sự kiện như trong Hình 2-2. Đối với mỗi bước sự kiện ban đầu, trình hòa giải sự kiện sẽ tạo một sự kiện xử lý (ví dụ: thay đổi địa chỉ, tính toán lại báo giá, v.v.), gửi sự kiện xử lý đó đến kênh sự kiện và đợi sự kiện xử lý được xử lý bởi bộ xử lý sự kiện tương ứng (ví dụ: , quy trình khách hàng, quy trình báo giá, v.v.). Quá trình này tiếp tục cho đến khi tất cả các bước trong sự kiện ban đầu được xử lý. Thanh duy nhất phía trên các bước tính toán lại và cập nhật xác nhận quyền sở hữu trong trình hòa giải sự kiện cho biết rằng các bước này có thể được chạy cùng một lúc.

Cấu trúc liên kết môi giới

Cấu trúc liên kết của người môi giới khác với cấu trúc liên kết của người hòa giải ở chỗ không có người hòa giải sự kiện trung tâm; đúng hơn, luồng tin nhắn được phân phối trên các thành phần của bộ xử lý sự kiện theo kiểu giống như chuỗi thông qua một trình trung chuyển tin nhắn nhẹ (ví dụ: ActiveMQ, HornetQ, v.v.). Cấu trúc liên kết này hữu ích khi bạn có luồng xử lý sự kiện tương đối đơn giản và bạn không muốn (hoặc cần) việc điều phối sự kiện trung tâm.

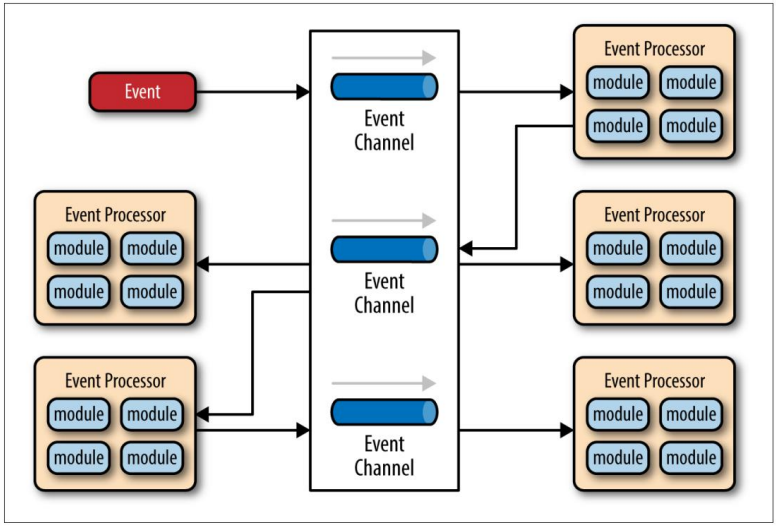
Có hai loại thành phần kiến trúc chính trong cấu trúc liên kết môi giới: thành phần môi giới và thành phần xử lý sự kiện. Thành phần môi giới có thể được tập trung hoặc liên kết và chứa tất cả các kênh sự kiện được sử dụng trong luồng sự kiện.

Các kênh sự kiện có trong thành phần môi giới có thể là hàng đợi tin nhắn, chủ đề tin nhắn hoặc kết hợp cả hai.



Hình 2-2. Ví dụ về cấu trúc liên kết hòa giải

Cấu trúc liên kết này được minh họa trong Hình 2-3. Như bạn có thể thấy từ sơ đồ, không có thành phần trung gian hòa giải sự kiện nào kiểm soát và điều phối sự kiện ban đầu; đúng hơn, mỗi thành phần bộ xử lý sự kiện chịu trách nhiệm xử lý một sự kiện và xuất bản một sự kiện mới cho biết hành động mà nó vừa thực hiện. Ví dụ: bộ xử lý sự kiện cân bằng danh mục cổ phiếu có thể nhận được sự kiện ban đầu được gọi là chia tách cổ phiếu. Dựa trên sự kiện ban đầu đó, bộ xử lý sự kiện có thể thực hiện một số việc tái cân bằng danh mục đầu tư, sau đó xuất bản một sự kiện mới cho nhà môi giới được gọi là danh mục tái cân bằng, sau đó sẽ được một bộ xử lý sự kiện khác chọn. Lưu ý rằng đôi khi một sự kiện được xuất bản bởi bộ xử lý sự kiện nhưng không được bất kỳ bộ xử lý sự kiện nào khác chọn. Điều này thường xảy ra khi bạn đang phát triển một ứng dụng hoặc cung cấp các chức năng và tiện ích mở rộng trong tương lai.

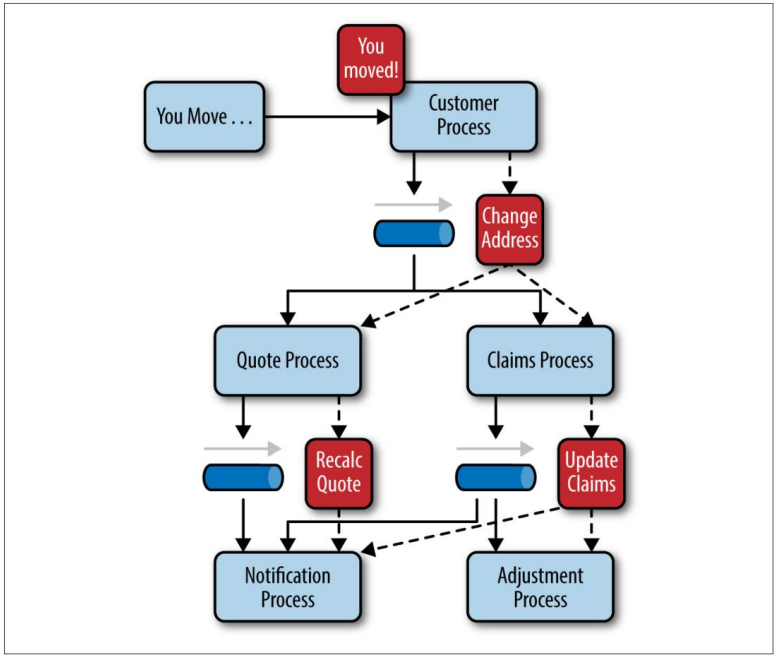


Hình 2-3. Cấu trúc liên kết môi giới kiến trúc hướng sự kiện

Để minh họa cách hoạt động của cấu trúc liên kết môi giới, chúng tôi sẽ sử dụng ví dụ tương tự như trong cấu trúc liên kết hòa giải (một người được bảo hiểm di chuyển). Do không có người hòa giải sự kiện trung tâm để nhận sự kiện ban đầu trong cấu trúc liên kết của nhà môi giới, nên thành phần quy trình khách hàng sẽ nhận sự kiện trực tiếp, thay đổi địa chỉ của khách hàng và gửi đi một sự kiện cho biết nó đã thay đổi địa chỉ của khách hàng (ví dụ: sự kiện thay đổi địa chỉ) .

Trong ví dụ này, có hai bộ xử lý sự kiện quan tâm đến sự kiện thay đổi địa chỉ: quy trình báo giá và quy trình yêu cầu bồi thường.

Thành phần bộ xử lý báo giá sẽ tính toán lại tỷ lệ bảo hiểm ô tô mới dựa trên sự thay đổi địa chỉ và xuất bản một sự kiện cho phần còn lại của hệ thống cho biết nó đã làm gì (ví dụ: tính toán lại sự kiện báo giá). Mặt khác, thành phần xử lý yêu cầu bồi thường nhận được cùng một sự kiện thay đổi địa chỉ, nhưng trong trường hợp này, nó cập nhật một yêu cầu bồi thường bảo hiểm chưa thanh toán và xuất bản một sự kiện lên hệ thống dưới dạng sự kiện cập nhật yêu cầu bồi thường. Sau đó, các sự kiện mới này được các thành phần xử lý sự kiện khác tiếp nhận và chuỗi sự kiện tiếp tục diễn ra trong hệ thống cho đến khi không còn sự kiện nào được xuất bản cho sự kiện bắt đầu cụ thể đó.



Hình 2-4. Ví dụ về cấu trúc liên kết của nhà môi giới

Như bạn có thể thấy trong Hình 2-4, cấu trúc liên kết của nhà môi giới là về chuỗi các sự kiện để thực hiện một chức năng kinh doanh. Cách tốt nhất để hiểu cấu trúc liên kết của nhà môi giới là coi nó như một cuộc đua tiếp sức.

Trong cuộc đua tiếp sức, người chạy cầm dùi cui và chạy một quãng đường nhất định, sau đó trao dùi cui cho người chạy tiếp theo, cứ như vậy xuống chuỗi cho đến khi người chạy cuối cùng vượt qua vạch đích. Trong các cuộc đua tiếp sức, khi một vận động viên bỏ gậy ra, cô ấy sẽ hoàn thành cuộc đua. Điều này cũng đúng với cấu trúc liên kết của nhà môi giới: một khi bộ xử lý sự kiện xử lý sự kiện, nó sẽ không còn liên quan đến việc xử lý sự kiện cụ thể đó nữa.

Cân nhắc

Mẫu kiến trúc hướng sự kiện là một mẫu tương đối phức tạp để triển khai, chủ yếu do tính chất phân tán không đồng bộ của nó.

Khi triển khai mẫu này, bạn phải giải quyết các vấn đề về kiến trúc phân tán khác nhau, chẳng hạn như tính khả dụng của quy trình từ xa, thiếu khả năng phản hồi và logic kết nối lại của nhà môi giới trong trường hợp nhà môi giới hoặc người hòa giải bị lỗi.

Một điều cần cân nhắc khi chọn mẫu kiến trúc này là việc thiếu các giao dịch nguyên tử cho một quy trình kinh doanh đơn lẻ. Bởi vì các thành phần của bộ xử lý sự kiện có tính tách rời và phân tán cao nên rất khó để duy trì một đơn vị công việc giao dịch giữa chúng. Vì lý do này, khi thiết kế ứng dụng của bạn bằng mẫu này, bạn phải liên tục suy nghĩ về những sự kiện nào có thể và không thể chạy độc lập và lập kế hoạch chi tiết cho các bộ xử lý sự kiện của bạn cho phù hợp. Nếu bạn thấy rằng bạn cần phân chia một đơn vị công việc cho các bộ xử lý sự kiện—tức là, nếu bạn đang sử dụng các bộ xử lý riêng biệt cho một thứ gì đó lẽ ra phải là một giao dịch không phân chia—thì đây có thể không phải là mẫu phù hợp cho ứng dụng của bạn.

Có lẽ một trong những khía cạnh khó khăn nhất của mẫu kiến trúc hướng sự kiện là việc tạo, duy trì và quản trị các hợp đồng thành phần bộ xử lý sự kiện. Mỗi sự kiện thường có một hợp đồng cụ thể được liên kết với nó (ví dụ: các giá trị dữ liệu và định dạng dữ liệu được chuyển đến bộ xử lý sự kiện). Điều cực kỳ quan trọng là khi sử dụng mẫu này để giải quyết định dạng dữ liệu tiêu chuẩn (ví dụ: XML, JSON, Đối tượng Java, v.v.) và thiết lập chính sách tạo phiên bản hợp đồng ngay từ đầu.

Phân tích mẫu

Bảng sau đây chứa xếp hạng và phân tích các đặc điểm kiến trúc chung cho mẫu kiến trúc hướng sự kiện.
Xếp hạng cho từng đặc điểm dựa trên xu hướng tự nhiên đối với đặc điểm đó như một khả năng dựa trên việc triển khai mẫu điển hình, cũng như mục đích chung của mẫu. Để so sánh song song mối liên hệ của mẫu này với các mẫu khác trong báo cáo này, vui lòng tham khảo Phụ lục A ở cuối báo cáo này.

Đánh giá mức độ

linh hoạt tổng

thể : Phân tích cao: Tính linh hoạt tổng thể là khả năng phản ứng nhanh chóng với môi trường thay đổi liên tục. Do các thành phần của bộ xử lý sự kiện có mục đích duy nhất và tách biệt hoàn toàn với các thành phần của bộ xử lý sự kiện khác nên các thay đổi thường được tách biệt với một hoặc một vài bộ xử lý sự kiện và có thể được thực hiện nhanh chóng mà không ảnh hưởng đến các thành phần khác.

Dễ triển khai Xếp hạng :

Phân tích cao:

Nhìn chung, mẫu này tương đối dễ triển khai do tính chất tách rời của các thành phần xử lý sự kiện. Cấu trúc liên kết môi giới có xu hướng dễ triển khai hơn cấu trúc liên kết trung gian, chủ yếu là do thành phần trung gian sự kiện được kết hợp chặt chẽ với bộ xử lý sự kiện: một thay đổi trong thành phần bộ xử lý sự kiện cũng có thể yêu cầu thay đổi trong bộ trung gian sự kiện, yêu cầu cả hai. được triển khai cho bất kỳ thay đổi nào.

Xếp hạng khả

năng kiểm

thử : Phân tích thấp: Mặc dù việc kiểm thử đơn vị riêng lẻ không quá khó nhưng nó yêu cầu một số loại ứng dụng khách hoặc công cụ kiểm thử chuyên dụng để tạo sự kiện. Việc kiểm tra cũng phức tạp do tính chất không đồng bộ của mẫu này.

Xếp hạng hiệu

suất : Phân

tích cao: Mặc dù chắc chắn có thể triển khai kiến trúc hướng sự kiện nhưng kiến trúc này không hoạt động tốt do tất cả cơ sở hạ tầng nhắn tin có liên quan, nhưng nhìn chung, mẫu này đạt được hiệu suất cao nhờ khả năng không đồng bộ của nó; nói cách khác, khả năng thực hiện các hoạt động không đồng bộ song song, tách rời sẽ vượt xa chi phí xếp hàng và loại bỏ các tin nhắn.

Xếp hạng khả

năng mở rộng :

Phân tích cao: Khả năng mở rộng đạt được một cách tự nhiên trong mẫu này thông qua các bộ xử lý sự kiện có tính độc lập cao và tách rời. Mỗi bộ xử lý sự kiện có thể được điều chỉnh tỷ lệ riêng biệt, cho phép khả năng mở rộng chi tiết.

Dễ phát triển Xếp hạng:

Phân tích

thấp: Quá trình phát triển có thể hơi phức tạp do tính chất không đồng bộ của mẫu cũng như việc tạo hợp đồng và nhu cầu về các điều kiện xử lý lỗi nâng cao hơn trong mã dành cho các bộ xử lý sự kiện không phản hồi và trình môi giới không phản hồi.

CHƯƠNG 3

Kiến trúc vi hạt nhân

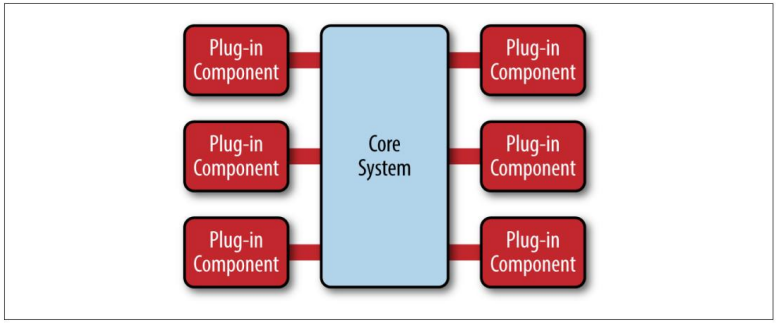
Mẫu kiến trúc vi nhân (đôi khi được gọi là mẫu kiến trúc plug-in) là một mẫu tự nhiên để triển khai các ứng dụng dựa trên sản phẩm. Ứng dụng dựa trên sản phẩm là ứng dụng được đóng gói và cung cấp để tải xuống ở các phiên bản dưới dạng sản phẩm thông thường của bên thứ ba. Tuy nhiên, nhiều công ty cũng phát triển và phát hành các ứng dụng kinh doanh nội bộ của họ như các sản phẩm phần mềm, hoàn chỉnh với các phiên bản, ghi chú phát hành và các tính năng có thể cắm được. Đây cũng là sự phù hợp tự nhiên cho mẫu này. Mẫu kiến trúc vi nhân cho phép bạn thêm các tính năng ứng dụng bổ sung dưới dạng plug-in vào ứng dụng cốt lõi, cung cấp khả năng mở rộng cũng như khả năng phân tách và cách ly tính năng.

Mô tả mẫu

Mẫu kiến trúc vi nhân bao gồm hai loại thành phần kiến trúc: hệ thống lõi và các mô-đun trình cắm thêm. Logic ứng dụng được phân chia giữa các mô-đun plug-in độc lập và hệ thống lõi cơ bản, mang lại khả năng mở rộng, linh hoạt và tách biệt các tính năng ứng dụng cũng như logic xử lý tùy chỉnh. Hình 3-1 minh họa mẫu kiến trúc vi nhân cơ bản.

Hệ thống cốt lõi của mẫu kiến trúc vi nhân theo truyền thống chỉ chứa chức năng tối thiểu cần thiết để vận hành hệ thống. Nhiều hệ điều hành triển khai mẫu kiến trúc vi nhân, do đó có nguồn gốc tên gọi mẫu này.
Từ góc độ ứng dụng kinh doanh, hệ thống cốt lõi thường

được định nghĩa là logic nghiệp vụ chung không có mã tùy chỉnh cho các trường hợp đặc biệt, quy tắc đặc biệt hoặc xử lý có điều kiện phức tạp.



Hình 3-1. Mô hình kiến trúc vi hạt nhân

Các mô-đun plug-in là các thành phần độc lập, độc lập, chứa quy trình xử lý chuyên biệt, các tính năng bổ sung và mã tùy chỉnh nhằm nâng cao hoặc mở rộng hệ thống cốt lõi nhằm tạo ra các khả năng kinh doanh bổ sung. Nói chung, các mô-đun trình cắm thêm phải độc lập với các mô-đun trình cắm thêm khác, nhưng bạn chắc chắn có thể thiết kế các trình cắm thêm yêu cầu phải có các trình cắm thêm khác. Dù bằng cách nào, điều quan trọng là giữ liên lạc giữa các plug-in ở mức tối thiểu để tránh các vấn đề phụ thuộc.

Hệ thống cốt lõi cần biết những mô-đun plug-in nào có sẵn và cách truy cập chúng. Một cách phổ biến để thực hiện điều này là thông qua một số loại sổ đăng ký plug-in. Sổ đăng ký này chứa thông tin về từng mô-đun trình cắm, bao gồm những thông tin như tên, hợp đồng dữ liệu và chi tiết giao thức truy cập từ xa (tùy thuộc vào cách trình cắm được kết nối với hệ thống lõi). Ví dụ: một plug-in dành cho phần mềm thuế gắn cờ các mục kiểm tra thuế có rủi ro cao có thể có mục đăng ký chứa tên dịch vụ (AuditChecker), hợp đồng dữ liệu (dữ liệu đầu vào và dữ liệu đầu ra) và hợp đồng định dạng (XML). Nó cũng có thể chứa WSDL (Ngôn ngữ định nghĩa dịch vụ web) nếu trình cắm thêm được truy cập thông qua SOAP.

Các mô-đun plug-in có thể được kết nối với hệ thống lõi thông qua nhiều cách khác nhau, bao gồm OSGi (sáng kiến cổng dịch vụ mở), nhắn tin, dịch vụ web hoặc thậm chí liên kết điểm-điểm trực tiếp (tức là khởi tạo đối tượng). Loại kết nối bạn sử dụng tùy thuộc vào loại ứng dụng bạn đang xây dựng (ứng dụng sản phẩm nhỏ hoặc ứng dụng doanh nghiệp lớn) và nhu cầu cụ thể của bạn (ví dụ: triển khai một lần hoặc loại bỏ

tri ân triển khai). Bản thân mẫu kiến trúc không chỉ định bất kỳ chi tiết triển khai nào trong số này, chỉ có điều các mô-đun trình cắm thêm phải độc lập với nhau.

Các hợp đồng giữa các mô-đun plug-in và hệ thống cốt lõi có thể có phạm vi từ hợp đồng tiêu chuẩn đến hợp đồng tùy chỉnh. Hợp đồng tùy chỉnh thường được tìm thấy trong các tình huống trong đó các thành phần plug-in được phát triển bởi bên thứ ba mà bạn không có quyền kiểm soát hợp đồng được sử dụng bởi plug-in. Trong những trường hợp như vậy, thông thường bạn nên tạo một bộ chuyển đổi giữa liên hệ trình cắm thêm và hợp đồng tiêu chuẩn của mình để hệ thống cốt lõi không cần mã chuyên dụng cho từng trình cắm thêm. Khi tạo các hợp đồng tiêu chuẩn (thường được triển khai thông qua XML hoặc Bản đồ Java), điều quan trọng cần nhớ là tạo chiến lược tạo phiên bản ngay từ đầu.

Ví dụ về mẫu

Có lẽ ví dụ điển hình nhất về kiến trúc vi nhân là IDE Eclipse. Việc tải xuống sản phẩm Eclipse cơ bản sẽ cung cấp cho bạn nhiều hơn một trình soạn thảo ưa thích. Tuy nhiên, khi bạn bắt đầu thêm plugin, nó sẽ trở thành một sản phẩm hữu ích và có khả năng tùy chỉnh cao.
Trình duyệt Internet là một ví dụ về sản phẩm phổ biến khác sử dụng kiến trúc vi nhân: trình xem và các phần bổ trợ khác bổ sung thêm các khả năng bổ sung không có trong trình duyệt cơ bản (tức là hệ thống lõi).

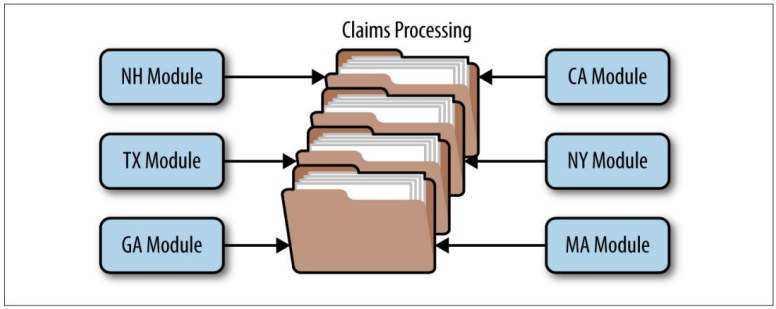
Các ví dụ là vô tận đối với phần mềm dựa trên sản phẩm, nhưng còn các ứng dụng kinh doanh lớn thì sao? Kiến trúc vi nhân cũng áp dụng cho những tình huống này. Để minh họa điểm này, chúng ta hãy sử dụng một ví dụ khác về công ty bảo hiểm, nhưng lần này là một ví dụ liên quan đến việc xử lý yêu cầu bồi thường bảo hiểm.

Xử lý yêu cầu bồi thường là một quá trình rất phức tạp. Mỗi tiểu bang có những quy tắc và quy định khác nhau về những gì được phép và không được phép trong yêu cầu bồi thường bảo hiểm. Ví dụ: một số tiểu bang cho phép thay kính chắn gió miễn phí nếu kính chắn gió của bạn bị đá làm hỏng, trong khi các tiểu bang khác thì không. Điều này tạo ra một tập hợp các điều kiện gần như vô hạn cho quy trình yêu cầu bồi thường tiêu chuẩn.

Không có gì ngạc nhiên khi hầu hết các ứng dụng yêu cầu bồi thường bảo hiểm đều tận dụng các công cụ quy tắc lớn và phức tạp để xử lý phần lớn sự phức tạp này. Tuy nhiên, các công cụ quy tắc này có thể phát triển thành một quả bóng bùn lớn phức tạp, trong đó việc thay đổi một quy tắc sẽ tác động đến các quy tắc khác hoặc tạo ra một quy tắc.

thay đổi quy tắc đơn giản đòi hỏi một đội ngũ các nhà phân tích, nhà phát triển và người thử nghiệm. Sử dụng mẫu kiến trúc vi nhân có thể giải quyết được nhiều vấn đề này.

Chồng các thư mục mà bạn nhìn thấy trong Hình 3-2 thể hiện hệ thống cốt lõi để xử lý khiếu nại. Nó chứa logic kinh doanh cơ bản mà công ty bảo hiểm yêu cầu để xử lý yêu cầu bồi thường, ngoại trừ việc không có bất kỳ xử lý tùy chỉnh nào. Mỗi mô-đun trình cắm thêm chứa các quy tắc cụ thể cho trạng thái đó. Trong ví dụ này, các mô-đun plug-in có thể được triển khai bằng cách sử dụng mã nguồn tùy chỉnh hoặc các phiên bản công cụ quy tắc riêng biệt.
Bất kể việc triển khai như thế nào, điểm mấu chốt là các quy tắc và quy trình xử lý theo từng trạng thái tách biệt với hệ thống xác nhận quyền sở hữu cốt lõi và có thể được thêm, xóa và thay đổi mà ít hoặc không ảnh hưởng đến phần còn lại của hệ thống cốt lõi hoặc các mô-đun bổ trợ khác .



Hình 3-2. Ví dụ về kiến trúc vi hạt nhân

Cân nhắc

Một điều tuyệt vời về mẫu kiến trúc vi nhân là nó có thể được nhúng hoặc sử dụng như một phần của mẫu kiến trúc khác.
Ví dụ: nếu mẫu này giải quyết được một vấn đề cụ thể mà bạn gặp phải với một khu vực dễ thay đổi cụ thể của ứng dụng, thì bạn có thể thấy rằng mình không thể triển khai toàn bộ kiến trúc bằng cách sử dụng mẫu này. Trong trường hợp này, bạn có thể nhúng mẫu kiến trúc microservices vào một mẫu khác mà bạn đang sử dụng (ví dụ: kiến trúc phân lớp). Tương tự, các thành phần xử lý sự kiện được mô tả trong phần trước về kiến trúc hướng sự kiện có thể được triển khai bằng cách sử dụng mẫu kiến trúc vi dịch vụ.

Mẫu kiến trúc microservices cung cấp sự hỗ trợ tuyệt vời cho thiết kế tiến hóa và phát triển gia tăng. Trước tiên, bạn có thể tạo ra một hệ thống lõi vững chắc và khi ứng dụng phát triển thì

về mặt tinh thần, hãy thêm các tính năng và chức năng mà không cần phải thực hiện những thay đổi đáng kể đối với hệ thống cốt lõi.

Đối với các ứng dụng dựa trên sản phẩm, mẫu kiến trúc vi nhân phải luôn là lựa chọn đầu tiên của bạn làm kiến trúc khởi đầu, đặc biệt đối với những sản phẩm mà bạn sẽ phát hành các tính năng bổ sung theo thời gian và muốn kiểm soát xem người dùng nào sẽ nhận được tính năng nào. Nếu theo thời gian, bạn nhận thấy mẫu đó không đáp ứng tất cả các yêu cầu của mình, bạn luôn có thể cấu trúc lại ứng dụng của mình sang một mẫu kiến trúc khác phù hợp hơn với các yêu cầu cụ thể của bạn.

Phân tích mẫu

Bảng sau đây chứa xếp hạng và phân tích các đặc điểm kiến trúc chung cho mẫu kiến trúc vi nhân.
Xếp hạng cho từng đặc điểm dựa trên xu hướng tự nhiên đối với đặc điểm đó như một khả năng dựa trên việc triển khai mẫu điển hình, cũng như mục đích chung của mẫu. Để so sánh song song mối liên hệ của mẫu này với các mẫu khác trong báo cáo này, vui lòng tham khảo Phụ lục A ở cuối báo cáo này.

Đánh giá mức độ
: linh hoạt
tổng thể : Phân tích cao: Tính linh hoạt tổng thể là khả năng phản ứng nhanh chóng với môi trường thay đổi liên tục. Những thay đổi phần lớn có thể được tách biệt và triển khai nhanh chóng thông qua các mô-đun plug-in được liên kết lỏng lẻo. Nhìn chung, hệ thống cốt lõi của hầu hết các kiến trúc vi nhân có xu hướng trở nên ổn định nhanh chóng và do đó khá mạnh mẽ và yêu cầu ít thay đổi theo thời gian.

Dễ triển khai Xếp
: hạng : Phân
tích cao: Tùy thuộc vào cách triển khai mẫu, các mô-đun plug-in có thể được thêm động vào hệ thống lõi trong thời gian chạy (ví dụ: triển khai nóng), giảm thiểu thời gian ngừng hoạt động trong quá trình triển khai.

Xếp hạng khả

năng kiểm tra :

Phân tích cao: Các mô-đun plug-in có thể được kiểm tra riêng biệt và có thể dễ dàng được hệ thống lõi mô phỏng để chứng minh hoặc tạo nguyên mẫu cho một tính năng cụ thể mà không có hoặc có ít hoặc không có thay đổi nào đối với hệ thống lõi.

Xếp hạng hiệu

suất : Phân tích

cao: Mặc dù mẫu vi nhân không tự nhiên phù hợp với các ứng dụng hiệu suất cao, nhưng nhìn chung, hầu hết các ứng dụng được xây dựng bằng mẫu kiến trúc vi nhân đều hoạt động tốt vì bạn có thể tùy chỉnh và hợp lý hóa các ứng dụng để chỉ bao gồm những tính năng bạn cần . Máy chủ ứng dụng JBoss là một ví dụ điển hình về điều này: với kiến trúc plug-in của nó, bạn có thể cắt giảm máy chủ ứng dụng xuống chỉ còn những tính năng bạn cần, loại bỏ các tính năng đắt tiền không được sử dụng như truy cập từ xa, nhắn tin và bộ nhớ đệm tiêu tốn bộ nhớ , CPU và các luồng và làm chậm máy chủ ứng dụng.

Xếp hạng khả

năng mở rộng :

Phân tích thấp: Bởi vì hầu hết việc triển khai kiến trúc vi nhân đều dựa trên sản phẩm và thường có kích thước nhỏ hơn nên chúng được triển khai dưới dạng các đơn vị đơn lẻ và do đó khả năng mở rộng không cao.

Tùy thuộc vào cách bạn triển khai các mô-đun plug-in, đôi khi bạn có thể cung cấp khả năng mở rộng ở cấp tính năng plug-in, nhưng nhìn chung mẫu này không được biết đến để tạo ra các ứng dụng có khả năng mở rộng cao.

Dễ phát triển Xếp hạng:

Phân tích thấp:

Kiến trúc vi nhân yêu cầu thiết kế chu đáo và quản lý hợp đồng, khiến việc triển khai khá phức tạp. Phiên bản hợp đồng, đăng ký phần bổ trợ nội bộ, mức độ chi tiết của phần bổ trợ và nhiều lựa chọn sẵn có cho kết nối phần bổ trợ đều góp phần tạo nên sự phức tạp liên quan đến việc triển khai mẫu này.

CHƯƠNG 4

Mẫu kiến trúc microservice

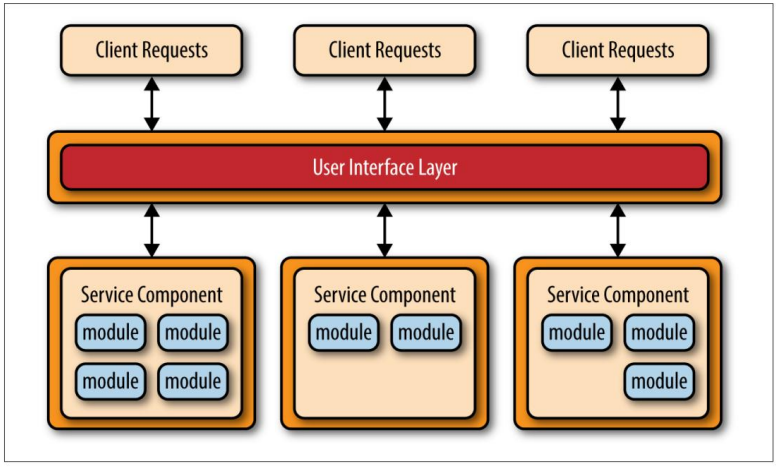
Mô hình kiến trúc microservices đang nhanh chóng có chỗ đứng trong ngành như một giải pháp thay thế khả thi cho các ứng dụng nguyên khối và kiến trúc hướng dịch vụ. Bởi vì mẫu kiến trúc này vẫn đang phát triển nên có rất nhiều nhầm lẫn trong ngành về nội dung của mẫu này và cách nó được triển khai. Phần này của báo cáo sẽ cung cấp cho bạn các khái niệm chính và kiến thức nền tảng cần thiết để hiểu lợi ích (và sự cân bằng) của mẫu kiến trúc quan trọng này và liệu nó có phải là mẫu phù hợp cho ứng dụng của bạn hay không.

Mô tả mẫu

Bất kể cấu trúc liên kết hoặc phong cách triển khai mà bạn chọn, có một số khái niệm cốt lõi chung áp dụng cho mẫu kiến trúc chung. Khái niệm đầu tiên trong số này là khái niệm về các đơn vị được triển khai riêng biệt. Như minh họa trong Hình 4-1, mỗi thành phần của kiến trúc vi dịch vụ được triển khai dưới dạng một đơn vị riêng biệt, cho phép triển khai dễ dàng hơn thông qua quy trình phân phối hiệu quả và hợp lý, tăng khả năng mở rộng cũng như mức độ tách rời ứng dụng và thành phần cao trong ứng dụng của bạn.

Có lẽ khái niệm quan trọng nhất cần hiểu với mẫu này là khái niệm về thành phần dịch vụ. Thay vì nghĩ về các dịch vụ trong kiến trúc microservice, tốt hơn nên nghĩ về các thành phần dịch vụ, chúng có thể khác nhau về mức độ chi tiết từ một mô-đun đơn lẻ đến một phần lớn của ứng dụng. Các thành phần dịch vụ chứa một hoặc nhiều mô-đun (ví dụ: các lớp Java) đại diện cho một trong hai

một chức năng có mục đích duy nhất (ví dụ: cung cấp thời tiết cho một thành phố hoặc thị trấn cụ thể) hoặc một phần độc lập của ứng dụng kinh doanh lớn (ví dụ: vị trí giao dịch chứng khoán hoặc xác định tỷ lệ bảo hiểm ô tô). Thiết kế mức độ chi tiết của thành phần dịch vụ phù hợp là một trong những thách thức lớn nhất trong kiến trúc microservice. Thử thách này sẽ được thảo luận chi tiết hơn trong tiểu mục điều phối thành phần dịch vụ sau đây.



Hình 4-1. Mẫu kiến trúc Microservices cơ bản

Một khái niệm quan trọng khác trong mẫu kiến trúc microservices là nó là kiến trúc phân tán, nghĩa là tất cả các thành phần trong kiến trúc được tách rời hoàn toàn với nhau và được truy cập thông qua một số loại giao thức truy cập từ xa (ví dụ: JMS, AMQP, REST, XÀ PHÒNG, RMI, v.v.). Bản chất phân tán của mẫu kiến trúc này là cách nó đạt được một số đặc điểm triển khai và khả năng mở rộng vượt trội.

Một trong những điều thú vị về kiến trúc microservice là nó phát triển từ các vấn đề liên quan đến các mẫu kiến trúc phổ biến khác, thay vì được tạo ra như một giải pháp chờ sự cố xảy ra. Phong cách kiến trúc microservice phát triển một cách tự nhiên từ hai nguồn chính: các ứng dụng nguyên khối được phát triển bằng mẫu kiến trúc phân lớp và các ứng dụng phân tán được phát triển thông qua mẫu kiến trúc hướng dịch vụ.

Con đường phát triển từ các ứng dụng nguyên khối đến phong cách kiến trúc vi dịch vụ được thúc đẩy chủ yếu thông qua sự phát triển của phân phối liên tục, khái niệm về một quy trình liên tục.

Quy trình triển khai từ phát triển đến sản xuất giúp đơn giản hóa việc triển khai ứng dụng. Các ứng dụng nguyên khối thường bao gồm các thành phần được liên kết chặt chẽ là một phần của một đơn vị có thể triển khai duy nhất, khiến cho việc thay đổi, kiểm tra và triển khai ứng dụng trở nên cồng kềnh và khó khăn (do đó, sự gia tăng của các chu kỳ "triển khai hàng tháng" thường thấy ở hầu hết các ứng dụng lớn) cửa hàng CNTT). Những yếu tố này thường dẫn đến các ứng dụng dễ hỏng, dễ bị hỏng mỗi khi triển khai nội dung mới. Mẫu kiến trúc microservices giải quyết những vấn đề này bằng cách tách ứng dụng thành nhiều đơn vị có thể triển khai (các thành phần dịch vụ) có thể được phát triển, thử nghiệm và triển khai riêng lẻ, độc lập với các thành phần dịch vụ khác.

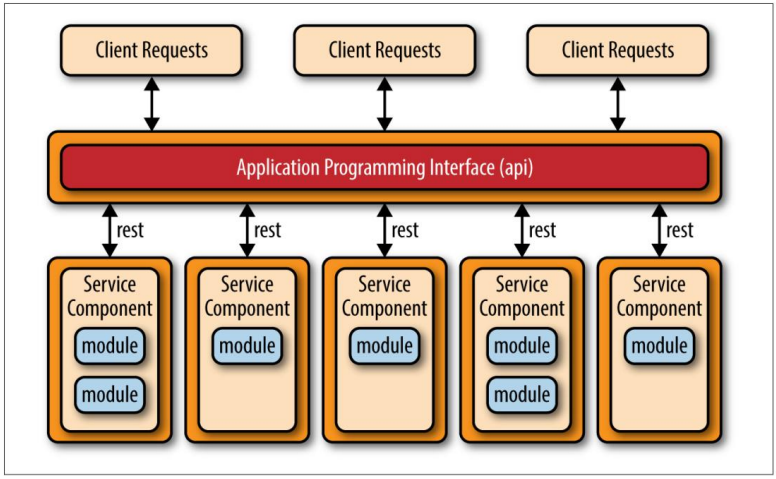
Con đường tiến hóa khác dẫn đến mẫu kiến trúc vi dịch vụ là từ các vấn đề được tìm thấy với các ứng dụng triển khai mẫu kiến trúc hướng dịch vụ (SOA). Mặc dù mẫu SOA rất mạnh mẽ và cung cấp mức độ trừu tượng vô song, khả năng kết nối không đồng nhất, điều phối dịch vụ và hứa hẹn điều chỉnh các mục tiêu kinh doanh phù hợp với khả năng CNTT, tuy nhiên, nó vẫn phức tạp, tốn kém, phổ biến, khó hiểu và khó thực hiện, và thường là quá mức cần thiết cho hầu hết các ứng dụng. Phong cách kiến trúc microservices giải quyết sự phức tạp này bằng cách đơn giản hóa khái niệm dịch vụ, loại bỏ nhu cầu điều phối cũng như đơn giản hóa khả năng kết nối và quyền truy cập vào các thành phần dịch vụ.

Cấu trúc liên kết mẫu

Mặc dù có hàng chục cách để triển khai mẫu kiến trúc vi dịch vụ, nhưng có ba cấu trúc liên kết chính nổi bật và phổ biến nhất: cấu trúc liên kết dựa trên API REST, cấu trúc liên kết dựa trên ứng dụng REST và cấu trúc liên kết nhắn tin tập trung.

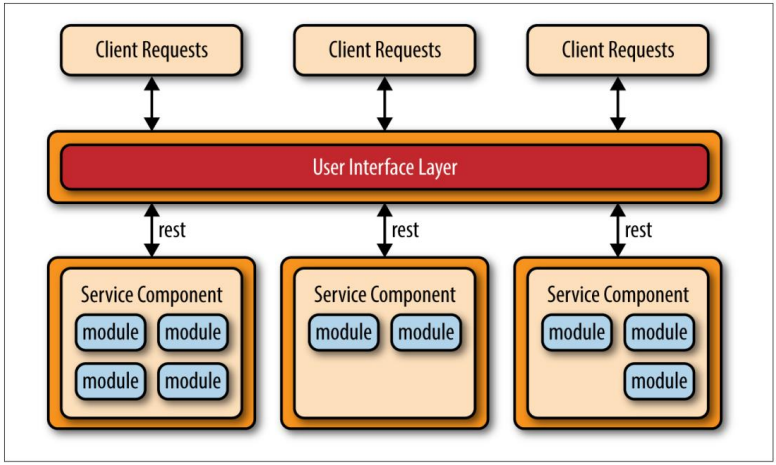
Cấu trúc liên kết dựa trên API REST rất hữu ích cho các trang web hiển thị các dịch vụ riêng lẻ, nhỏ, khép kín thông qua một số loại API (giao diện lập trình ứng dụng). Cấu trúc liên kết này, được minh họa trong Hình 4-2, bao gồm các thành phần dịch vụ rất chi tiết (do đó có tên là microservices) chứa một hoặc hai mô-đun thực hiện các chức năng kinh doanh cụ thể độc lập với phần còn lại của dịch vụ. Trong cấu trúc liên kết này, các thành phần dịch vụ chi tiết này thường được truy cập bằng giao diện dựa trên REST được triển khai thông qua lớp API dựa trên web được triển khai riêng.
Ví dụ về cấu trúc liên kết này bao gồm một số cấu trúc đơn phổ biến

mục đích dịch vụ web RESTful dựa trên đám mây được tìm thấy bởi Yahoo, Google và Amazon.



Hình 4-2. Cấu trúc liên kết dựa trên API REST

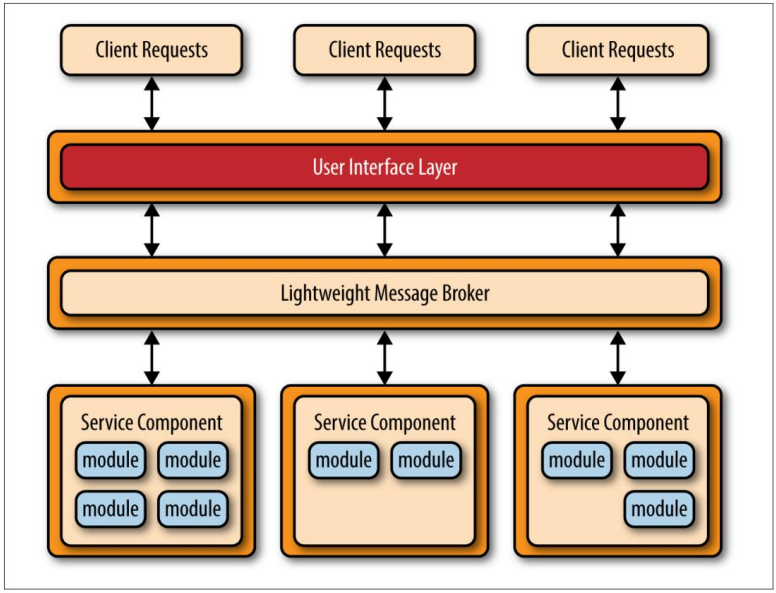
Cấu trúc liên kết dựa trên REST của ứng dụng khác với cách tiếp cận dựa trên API REST ở chỗ các yêu cầu của máy khách được nhận thông qua màn hình ứng dụng doanh nghiệp dựa trên web hoặc máy khách truyền thống thay vì thông qua một lớp API đơn giản. Như minh họa trong Hình 4-3, lớp giao diện người dùng của ứng dụng được triển khai dưới dạng một ứng dụng web riêng biệt truy cập từ xa vào các thành phần dịch vụ được triển khai riêng biệt (chức năng nghiệp vụ) thông qua các giao diện dựa trên REST đơn giản. Các thành phần dịch vụ trong cấu trúc liên kết này khác với các thành phần dịch vụ trong cấu trúc liên kết dựa trên API-REST ở chỗ các thành phần dịch vụ này có xu hướng lớn hơn, chi tiết hơn và chiếm một phần nhỏ trong ứng dụng kinh doanh tổng thể thay vì chi tiết đơn lẻ. dịch vụ hành động. Cấu trúc liên kết này phổ biến cho các ứng dụng kinh doanh vừa và nhỏ có mức độ phức tạp tương đối thấp.



Hình 4-3. Cấu trúc liên kết dựa trên REST của ứng dụng

Một cách tiếp cận phổ biến khác trong mẫu kiến trúc microservice là cấu trúc liên kết nhắn tin tập trung. Cấu trúc liên kết này (được minh họa trong Hình 4-4) tương tự như cấu trúc liên kết dựa trên REST của ứng dụng trước đó ngoại trừ việc thay vì sử dụng REST để truy cập từ xa, cấu trúc liên kết này sử dụng một trình trung chuyển tin nhắn tập trung nhẹ (ví dụ: ActiveMQ, HornetQ, v.v.) . Điều cực kỳ quan trọng khi xem xét cấu trúc liên kết này là không nhầm lẫn nó với mẫu kiến trúc hướng dịch vụ hoặc coi nó là “SOA-Lite”. Trình môi giới thông báo nhẹ được tìm thấy trong cấu trúc liên kết này không thực hiện bất kỳ sự sắp xếp, chuyển đổi hoặc định tuyến phức tạp nào. ; đúng hơn, nó chỉ là một phương tiện nhẹ để truy cập các thành phần dịch vụ từ xa.

Cấu trúc liên kết nhắn tin tập trung thường được tìm thấy trong các ứng dụng kinh doanh lớn hơn hoặc các ứng dụng yêu cầu kiểm soát phức tạp hơn đối với lớp truyền tải giữa giao diện người dùng và các thành phần dịch vụ. Lợi ích của cấu trúc liên kết này so với cấu trúc liên kết dựa trên REST đơn giản đã thảo luận trước đây là cơ chế xếp hàng nâng cao, nhắn tin không đồng bộ, giám sát, xử lý lỗi cũng như khả năng mở rộng và cân bằng tải tổng thể tốt hơn. Điểm lỗi duy nhất và các vấn đề tắc nghẽn kiến trúc thường liên quan đến một nhà môi giới tập trung được giải quyết thông qua việc phân cụm nhà môi giới và liên kết nhà môi giới (tách một phiên bản nhà môi giới thành nhiều phiên bản nhà môi giới để phân chia tải thông lượng tin nhắn dựa trên các khu vực chức năng của hệ thống).



Hình 4-4. Cấu trúc liên kết nhắn tin tập trung

Tránh sự phụ thuộc và phối hợp

Một trong những thách thức chính của mẫu kiến trúc microservices là xác định mức độ chi tiết chính xác cho các thành phần dịch vụ. Nếu các thành phần dịch vụ quá thô, bạn có thể không nhận ra những lợi ích đi kèm với mẫu kiến trúc này (triển khai, khả năng mở rộng, khả năng kiểm tra và khớp nối lỏng lẻo). Tuy nhiên, các thành phần dịch vụ quá chi tiết sẽ dẫn đến các yêu cầu về điều phối dịch vụ, điều này sẽ nhanh chóng biến kiến trúc dịch vụ vi mô tinh gọn của bạn thành một kiến trúc nặng nề hướng đến dịch vụ, hoàn chỉnh với tất cả sự phức tạp, nhầm lẫn, chi phí và những thứ vớ vẩn thông thường. được tìm thấy với các ứng dụng dựa trên SOA.

Nếu bạn thấy mình cần sắp xếp các thành phần dịch vụ của mình từ bên trong giao diện người dùng hoặc lớp API của ứng dụng thì rất có thể các thành phần dịch vụ của bạn quá chi tiết. Tương tự, nếu bạn thấy mình cần thực hiện liên lạc giữa các dịch vụ giữa các thành phần dịch vụ để xử lý một yêu cầu duy nhất, rất có thể các thành phần dịch vụ của bạn quá chi tiết hoặc chúng không được phân vùng chính xác theo quan điểm chức năng kinh doanh.

Thay vào đó, giao tiếp giữa các dịch vụ có thể buộc các kết nối không mong muốn giữa các thành phần có thể được xử lý thông qua một

cơ sở dữ liệu được chia sẻ. Ví dụ: nếu một thành phần dịch vụ xử lý các đơn đặt hàng qua Internet cần thông tin khách hàng, nó có thể đi tới cơ sở dữ liệu để truy xuất dữ liệu cần thiết thay vì gọi chức năng trong thành phần dịch vụ khách hàng.

Cơ sở dữ liệu dùng chung có thể xử lý các nhu cầu thông tin, nhưng còn chức năng dùng chung thì sao? Nếu một thành phần dịch vụ cần chức năng có trong một thành phần dịch vụ khác hoặc chung cho tất cả các thành phần dịch vụ, đôi khi bạn có thể sao chép chức năng được chia sẻ giữa các thành phần dịch vụ (do đó vi phạm nguyên tắc DRY: không lặp lại chính mình). Đây là một thực tế khá phổ biến trong hầu hết các ứng dụng kinh doanh triển khai mẫu kiến trúc vi dịch vụ, loại bỏ sự dư thừa của việc lặp lại các phần nhỏ của logic nghiệp vụ nhằm giữ cho các thành phần dịch vụ độc lập và tách biệt việc triển khai chúng. Các lớp tiện ích nhỏ có thể rơi vào loại mã lặp lại này.

Nếu bạn thấy rằng bất kể mức độ chi tiết của thành phần dịch vụ bạn vẫn không thể tránh được việc phối hợp thành phần dịch vụ thì đó là một dấu hiệu tốt cho thấy đây có thể không phải là mẫu kiến trúc phù hợp cho ứng dụng của bạn. Do tính chất phân tán của mô hình này, rất khó để duy trì một đơn vị công việc giao dịch duy nhất trên (và giữa) các thành phần dịch vụ. Cách thực hành như vậy sẽ yêu cầu một số loại khung bồi thường giao dịch để khôi phục các giao dịch, điều này làm tăng thêm độ phức tạp đáng kể cho mẫu kiến trúc tương đối đơn giản và thanh lịch này.

Cân nhắc

Mẫu kiến trúc microservices giải quyết nhiều vấn đề phổ biến được tìm thấy trong cả ứng dụng nguyên khối cũng như kiến trúc hướng dịch vụ. Do các thành phần ứng dụng chính được chia thành các đơn vị nhỏ hơn, được triển khai riêng biệt nên các ứng dụng được xây dựng bằng mẫu kiến trúc microservice thường mạnh mẽ hơn, cung cấp khả năng mở rộng tốt hơn và có thể hỗ trợ phân phối liên tục dễ dàng hơn.

Một ưu điểm khác của mẫu này là nó cung cấp khả năng triển khai sản xuất theo thời gian thực, do đó giảm đáng kể nhu cầu triển khai sản xuất “bùng nổ” hàng tháng hoặc cuối tuần truyền thống. Vì sự thay đổi thường được tách biệt với các thành phần dịch vụ cụ thể nên chỉ những thành phần dịch vụ thay đổi mới cần được triển khai. Nếu bạn chỉ có một phiên bản duy nhất của một dịch vụ

Ngoài ra, bạn có thể viết mã chuyên biệt trong ứng dụng giao diện người dùng để phát hiện triển khai nóng đang hoạt động và chuyển hướng người dùng đến trang lỗi hoặc trang chờ. Ngoài ra, bạn có thể trao đổi nhiều phiên bản của một thành phần dịch vụ trong quá trình triển khai theo thời gian thực, cho phép tính khả dụng liên tục trong các chu kỳ triển khai (điều rất khó thực hiện với mẫu kiến trúc phân lớp).

Một điều cần cân nhắc cuối cùng là vì mẫu kiến trúc dịch vụ vi mô là kiến trúc phân tán nên nó có một số vấn đề phức tạp tương tự được tìm thấy trong mẫu kiến trúc hướng sự kiện, bao gồm tạo hợp đồng, bảo trì và quản lý. , tính khả dụng của hệ thống từ xa cũng như xác thực và ủy quyền truy cập từ xa.

Phân tích mẫu

Bảng sau đây chứa xếp hạng và phân tích các đặc điểm kiến trúc chung cho mẫu kiến trúc vi dịch vụ. Xếp hạng cho từng đặc điểm dựa trên xu hướng tự nhiên đối với đặc điểm đó như một khả năng dựa trên việc triển khai mẫu điển hình, cũng như mục đích chung của mẫu. Để so sánh song song mối liên hệ của mẫu này với các mẫu khác trong báo cáo này, vui lòng tham khảo Phụ lục A ở cuối báo cáo này.

Đánh giá mức độ

linh hoạt tổng

thể : Phân tích cao: Tính linh hoạt tổng thể là khả năng phản ứng nhanh chóng với môi trường thay đổi liên tục. Do khái niệm về các đơn vị được triển khai riêng biệt, thay đổi thường được tách biệt thành các thành phần dịch vụ riêng lẻ, cho phép triển khai nhanh chóng và dễ dàng.

Ngoài ra, việc xây dựng các ứng dụng sử dụng mẫu này có xu hướng được liên kết rất lỏng lẻo, điều này cũng giúp tạo điều kiện thuận lợi cho sự thay đổi.

Dễ triển khai Xếp

hạng : Phân

tích cao: Nhìn chung, mẫu này tương đối dễ triển khai do tính chất tách rời của các thành phần xử lý sự kiện. Cấu trúc liên kết môi giới có xu hướng dễ triển khai hơn cấu trúc liên kết trung gian, chủ yếu là do thành phần trung gian sự kiện được kết hợp chặt chẽ với bộ xử lý sự kiện: một thay đổi trong thành phần bộ xử lý sự kiện cũng có thể yêu cầu thay đổi trong

người hòa giải sự kiện, yêu cầu cả hai phải được triển khai cho bất kỳ thay đổi nào.

Xếp hạng khả năng kiểm thử : Phân tích cao: Do sự tách biệt và tách biệt của chức năng kinh doanh thành các ứng dụng độc lập, việc kiểm thử có thể được xác định phạm vi, cho phép thực hiện nhiều nỗ lực kiểm thử có mục tiêu hơn. Kiểm tra hồi quy cho một thành phần dịch vụ cụ thể dễ dàng và khả thi hơn nhiều so với kiểm tra hồi quy cho toàn bộ ứng dụng nguyên khối. Ngoài ra, do các thành phần dịch vụ trong mẫu này được kết nối lỏng lẻo, nên từ góc độ phát triển, việc thực hiện thay đổi làm hỏng phần khác của ứng dụng sẽ ít có khả năng xảy ra, giảm bớt gánh nặng kiểm thử khi phải kiểm tra toàn bộ ứng dụng. một thay đổi nhỏ.

Xếp hạng hiệu suất : Phân tích thấp: Mặc dù bạn có thể tạo các ứng dụng được triển khai từ mẫu này hoạt động rất tốt nhưng nhìn chung, mẫu này không tự nhiên phù hợp với các ứng dụng hiệu suất cao do tính chất phân tán của mẫu kiến trúc vi dịch vụ.

Xếp hạng khả năng mở rộng : Phân tích cao: Do ứng dụng được chia thành các đơn vị được triển khai riêng biệt nên mỗi thành phần dịch vụ có thể được điều chỉnh tỷ lệ riêng lẻ, cho phép tinh chỉnh tỷ lệ ứng dụng. Ví dụ: khu vực quản trị của ứng dụng giao dịch chứng khoán có thể không cần mở rộng quy mô do số lượng người dùng cho chức năng đó thấp, nhưng thành phần dịch vụ vị trí giao dịch có thể cần mở rộng quy mô do hầu hết các ứng dụng giao dịch cần thông lượng cao cho việc này. chức năng.

Dễ phát triển Xếp hạng: Phân tích cao: Vì chức năng được tách biệt thành các thành phần dịch vụ riêng biệt và riêng biệt nên việc phát triển trở nên dễ dàng hơn do phạm vi nhỏ hơn và biệt lập. Ít có khả năng nhà phát triển thực hiện thay đổi trong một thành phần dịch vụ có thể ảnh hưởng đến các thành phần dịch vụ khác, do đó làm giảm sự phối hợp cần thiết giữa các nhà phát triển hoặc nhóm phát triển.

CHƯƠNG 5

Kiến trúc dựa trên không gian

Hầu hết các ứng dụng kinh doanh dựa trên web đều tuân theo cùng một luồng yêu cầu chung: yêu cầu từ trình duyệt truy cập máy chủ web, sau đó đến máy chủ ứng dụng và cuối cùng là máy chủ cơ sở dữ liệu. Mặc dù mô hình này hoạt động hiệu quả đối với một nhóm nhỏ người dùng, nhưng các nút thắt cổ chai bắt đầu xuất hiện khi tải của người dùng tăng lên, đầu tiên là ở lớp máy chủ web, sau đó là ở lớp máy chủ ứng dụng và cuối cùng là ở lớp máy chủ cơ sở dữ liệu. Phản ứng thông thường đối với các tắc nghẽn dựa trên sự gia tăng tải của người dùng là mở rộng quy mô các máy chủ web. Điều này tương đối dễ dàng và không tốn kém, và đôi khi có tác dụng giải quyết các vấn đề tắc nghẽn. Tuy nhiên, trong hầu hết các trường hợp lượng người dùng cao, việc mở rộng lớp máy chủ web chỉ chuyển nút cổ chai xuống máy chủ ứng dụng. Máy chủ ứng dụng mở rộng quy mô có thể phức tạp và đắt tiền hơn máy chủ web và thường chỉ di chuyển nút cổ chai xuống máy chủ cơ sở dữ liệu, việc mở rộng quy mô thậm chí còn khó khăn và tốn kém hơn. Ngay cả khi bạn có thể mở rộng quy mô cơ sở dữ liệu, cuối cùng kết quả bạn nhận được là cấu trúc liên kết hình tam giác, với phần rộng nhất của hình tam giác là các máy chủ web (dễ mở rộng quy mô nhất) và phần nhỏ nhất là cơ sở dữ liệu (khó chia tỷ lệ nhất).

Trong bất kỳ ứng dụng có khối lượng lớn nào có lượng người dùng đồng thời cực lớn, cơ sở dữ liệu thường sẽ là yếu tố giới hạn cuối cùng về số lượng giao dịch bạn có thể xử lý đồng thời. Mặc dù các công nghệ bộ nhớ đệm và sản phẩm mở rộng cơ sở dữ liệu khác nhau giúp giải quyết những vấn đề này, nhưng thực tế là việc mở rộng quy mô một ứng dụng thông thường cho các tải cực lớn là một đề xuất rất khó khăn.

Mẫu kiến trúc dựa trên không gian được thiết kế đặc biệt để giải quyết và giải quyết các vấn đề về khả năng mở rộng và tính tương tranh. Nó cũng là một mẫu kiến trúc hữu ích cho các ứng dụng có lượng người dùng đồng thời thay đổi và không thể đoán trước. Giải quyết vấn đề về khả năng mở rộng cực đoan và có thể thay đổi về mặt kiến trúc thường là cách tiếp cận tốt hơn so với việc cố gắng mở rộng quy mô cơ sở dữ liệu hoặc trang bị thêm công nghệ bộ nhớ đệm thành một kiến trúc không thể mở rộng.

Mô tả mẫu

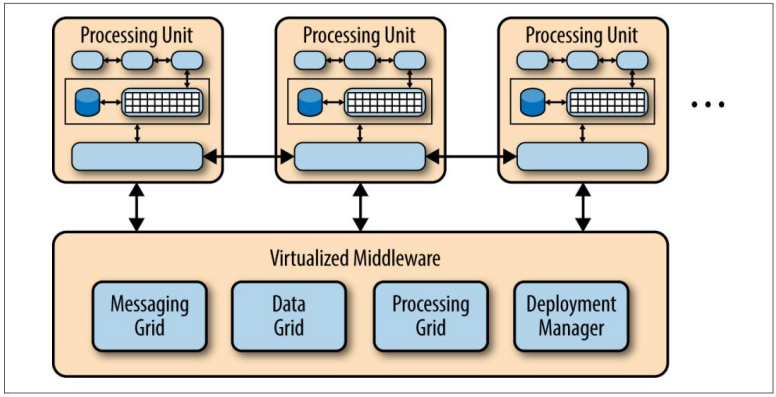
Mẫu dựa trên không gian (đôi khi còn được gọi là mẫu kiến trúc đám mây) giảm thiểu các yếu tố hạn chế khả năng mở rộng ứng dụng. Mẫu này lấy tên từ khái niệm không gian bộ dữ liệu, ý tưởng về bộ nhớ dùng chung phân tán. Khả năng mở rộng cao đạt được bằng cách loại bỏ ràng buộc cơ sở dữ liệu trung tâm và thay vào đó sử dụng lưới dữ liệu trong bộ nhớ được sao chép. Dữ liệu ứng dụng được lưu trong bộ nhớ và được sao chép giữa tất cả các đơn vị xử lý đang hoạt động. Các đơn vị xử lý có thể được khởi động và tắt một cách linh hoạt khi tải của người dùng tăng và giảm, từ đó giải quyết được khả năng mở rộng đa dạng. Do không có cơ sở dữ liệu trung tâm nên nút thắt cổ chai cơ sở dữ liệu sẽ được loại bỏ, mang lại khả năng mở rộng gần như vô hạn trong ứng dụng.

Hầu hết các ứng dụng phù hợp với mẫu này là các trang web tiêu chuẩn nhận được yêu cầu từ trình duyệt và thực hiện một số loại hành động. Một trang web đấu giá đấu thầu là một ví dụ điển hình về điều này. Trang web liên tục nhận được giá thầu từ người dùng internet thông qua yêu cầu của trình duyệt. Ứng dụng sẽ nhận được giá thầu cho một mặt hàng cụ thể, ghi lại giá thầu đó bằng dấu thời gian và cập nhật thông tin giá thầu mới nhất cho mặt hàng đó và gửi thông tin trở lại trình duyệt.

Có hai thành phần chính trong mẫu kiến trúc này: đơn vị xử lý và phần mềm trung gian ảo hóa. Hình 5-1 minh họa mẫu kiến trúc dựa trên không gian cơ bản và các thành phần kiến trúc chính của nó.

Thành phần đơn vị xử lý chứa các thành phần ứng dụng (hoặc các phần của thành phần ứng dụng). Điều này bao gồm các thành phần dựa trên web cũng như logic nghiệp vụ phụ trợ. Nội dung của đơn vị xử lý thay đổi tùy theo loại ứng dụng—các ứng dụng dựa trên web nhỏ hơn có thể sẽ được triển khai thành một đơn vị xử lý duy nhất, trong khi các ứng dụng lớn hơn có thể chia chức năng ứng dụng thành nhiều đơn vị xử lý dựa trên các khu vực chức năng của ứng dụng. Bộ xử lý thường

chứa các mô-đun ứng dụng, cùng với lưới dữ liệu trong bộ nhớ và kho lưu trữ liên tục không đồng bộ tùy chọn để chuyển đổi dự phòng. Nó cũng chứa một công cụ sao chép được phần mềm trung gian ảo hóa sử dụng để sao chép các thay đổi dữ liệu được thực hiện bởi một đơn vị xử lý sang các đơn vị xử lý đang hoạt động khác.



Hình 5-1. Mô hình kiến trúc dựa trên không gian

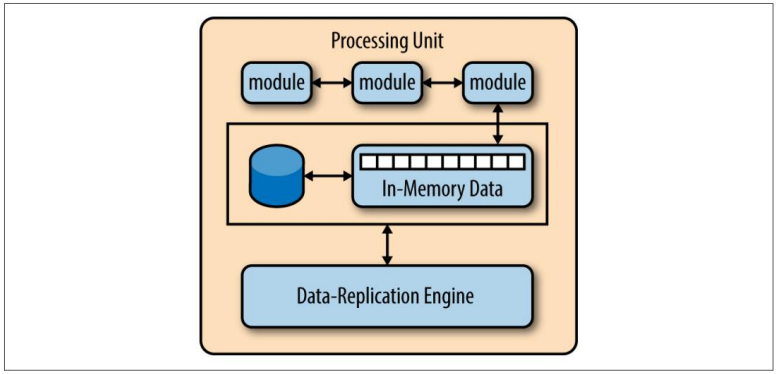
Thành phần phần mềm trung gian ảo hóa xử lý công việc dọn phòng và liên lạc. Nó chứa các thành phần kiểm soát các khía cạnh khác nhau của việc đồng bộ hóa dữ liệu và xử lý yêu cầu. Phần mềm trung gian ảo hóa bao gồm lưới nhắn tin, lưới dữ liệu, lưới xử lý và trình quản lý triển khai. Các thành phần này, được mô tả chi tiết trong phần tiếp theo, có thể được tùy chỉnh bằng văn bản hoặc mua dưới dạng sản phẩm của bên thứ ba.

Động lực học mẫu

Sự kỳ diệu của mẫu kiến trúc dựa trên không gian nằm ở các thành phần phần mềm trung gian được ảo hóa và lưới dữ liệu trong bộ nhớ có trong mỗi đơn vị xử lý. Hình 5-2 cho thấy kiến trúc đơn vị xử lý điển hình chứa các mô-đun ứng dụng, lưới dữ liệu trong bộ nhớ, kho lưu trữ lâu dài không đồng bộ tùy chọn để khắc phục sự cố và công cụ sao chép dữ liệu.

Phần mềm trung gian ảo hóa về cơ bản là bộ điều khiển kiến trúc và quản lý các yêu cầu, phiên, sao chép dữ liệu, xử lý yêu cầu phân tán và triển khai đơn vị quy trình. Có bốn thành phần kiến trúc chính trong phần mềm trung gian ảo hóa:

lưới nhắn tin, lưới dữ liệu, lưới xử lý và trình quản lý triển khai.



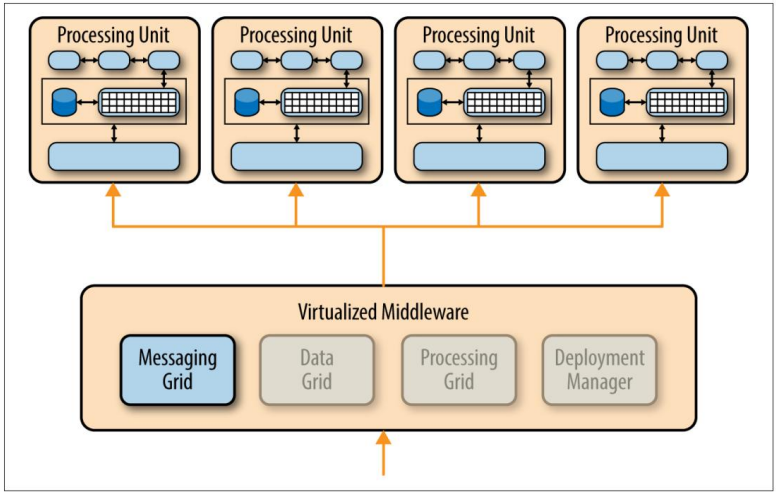
Hình 5-2. Thành phần đơn vị xử lý

Lưới nhắn tin Lưới

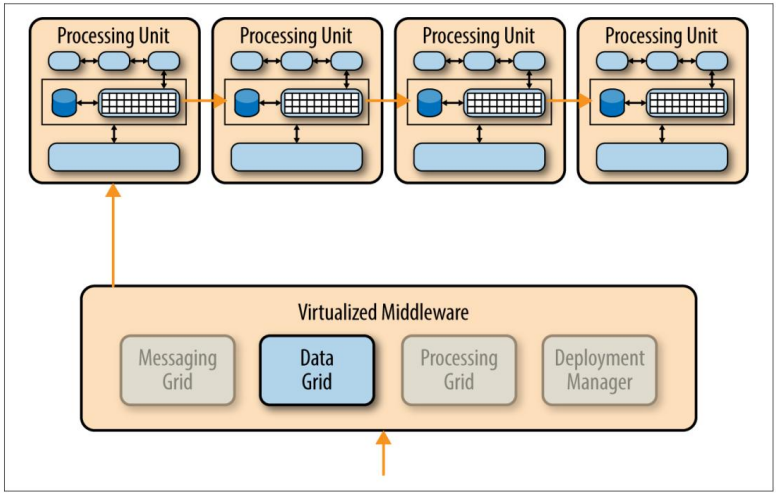
nhắn tin, được hiển thị trong Hình 5-3, quản lý yêu cầu đầu vào và thông tin phiên. Khi có yêu cầu đến thành phần phần mềm trung gian ảo hóa, thành phần lưới nhắn tin sẽ xác định thành phần xử lý đang hoạt động nào có sẵn để nhận yêu cầu và chuyển tiếp yêu cầu đến một trong các đơn vị xử lý đó. Độ phức tạp của lưới nhắn tin có thể dao động từ thuật toán quay vòng đơn giản đến thuật toán phức tạp hơn có sẵn tiếp theo để theo dõi yêu cầu nào đang được xử lý bởi đơn vị xử lý nào.

Lưới dữ liệu

Thành phần lưới dữ liệu có lẽ là thành phần quan trọng và cốt yếu nhất trong mẫu này. Lưới dữ liệu tương tác với công cụ sao chép dữ liệu trong mỗi đơn vị xử lý để quản lý việc sao chép dữ liệu giữa các đơn vị xử lý khi xảy ra cập nhật dữ liệu. Vì lưới nhắn tin có thể chuyển tiếp yêu cầu đến bất kỳ đơn vị xử lý nào có sẵn nên điều cần thiết là mỗi đơn vị xử lý phải chứa chính xác cùng một dữ liệu trong lưới dữ liệu trong bộ nhớ của nó. Mặc dù Hình 5-4 cho thấy sự sao chép dữ liệu đồng bộ giữa các đơn vị xử lý, nhưng trên thực tế, việc này được thực hiện song song không đồng bộ và rất nhanh chóng, đôi khi hoàn thành việc đồng bộ hóa dữ liệu chỉ trong vài micro giây (một phần triệu giây).



Hình 5-3. Thành phần lưới nhắn tin

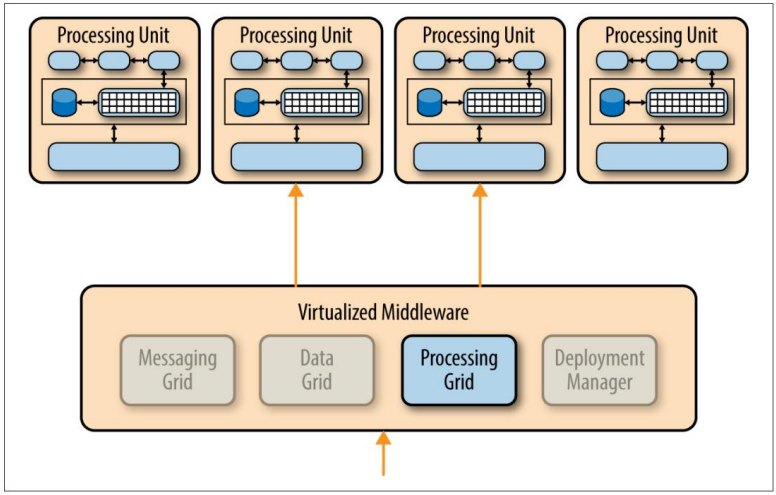


Hình 5-4. Thành phần lưới dữ liệu

Lưới xử lý Lưới xử

lý, được minh họa trong Hình 5-5, là một thành phần tùy chọn trong phần mềm trung gian ảo hóa quản lý việc xử lý yêu cầu phân tán khi có nhiều đơn vị xử lý, mỗi đơn vị xử lý một phần của ứng dụng. Nếu có yêu cầu yêu cầu phối hợp giữa các loại đơn vị xử lý (ví dụ: đơn vị xử lý đơn hàng và đơn vị xử lý khách hàng), thì đó là đơn vị xử lý.

lưới làm trung gian và điều phối yêu cầu giữa hai đơn vị xử lý đó.



Hình 5-5. Thành phần lưới xử lý

Trình quản lý triển khai

Thành phần trình quản lý triển khai quản lý việc khởi động và tắt động các đơn vị xử lý dựa trên các điều kiện tải. Thành phần này liên tục theo dõi thời gian phản hồi và tải của người dùng, đồng thời khởi động các bộ xử lý mới khi tải tăng và tắt các bộ xử lý khi tải giảm. Nó là một thành phần quan trọng để đạt được nhu cầu về khả năng mở rộng khác nhau trong một ứng dụng.

Cân nhắc

Mẫu kiến trúc dựa trên không gian là một mẫu phức tạp và tốn kém để thực hiện. Đó là một lựa chọn kiến trúc tốt cho các ứng dụng dựa trên web nhỏ hơn với tải thay đổi (ví dụ: các trang truyền thông xã hội, các trang đấu giá và đấu giá). Tuy nhiên, nó không phù hợp lắm với các ứng dụng cơ sở dữ liệu quan hệ quy mô lớn truyền thống với lượng dữ liệu vận hành lớn.

Mặc dù mẫu kiến trúc dựa trên không gian không yêu cầu kho dữ liệu tập trung, nhưng một kho dữ liệu thường được đưa vào để thực hiện tải lưới dữ liệu trong bộ nhớ ban đầu và duy trì các cập nhật dữ liệu không đồng bộ do các đơn vị xử lý thực hiện. Đó cũng là một cách phổ biến để tạo các phân vùng riêng biệt cách ly dễ bay hơi và được sử dụng rộng rãi

dữ liệu giao dịch từ dữ liệu không hoạt động, nhằm giảm dung lượng bộ nhớ của lưới dữ liệu trong bộ nhớ trong mỗi đơn vị xử lý.

Điều quan trọng cần lưu ý là mặc dù tên thay thế của mẫu này là kiến trúc dựa trên đám mây, nhưng các đơn vị xử lý (cũng như phần mềm trung gian ảo hóa) không nhất thiết phải nằm trên các dịch vụ được lưu trữ trên nền tảng đám mây hoặc PaaS (nền tảng như một dịch vụ). Nó có thể dễ dàng cư trú trên các máy chủ cục bộ, đó là một trong những lý do khiến tôi thích cái tên "kiến trúc dựa trên không gian" hơn.

Từ góc độ triển khai sản phẩm, bạn có thể triển khai nhiều thành phần kiến trúc trong mẫu này thông qua các sản phẩm của bên thứ ba như GemFire, JavaSpaces, GigaSpaces, IBM Object Grid, nCache và Oracle Coherence. Bởi vì việc triển khai mẫu này khác nhau rất nhiều về mặt chi phí và khả năng (đặc biệt là thời gian sao chép dữ liệu), với tư cách là một kiến trúc sư, trước tiên bạn nên thiết lập mục tiêu và nhu cầu cụ thể của mình là gì trước khi thực hiện bất kỳ lựa chọn sản phẩm nào.

Phân tích mẫu

Bảng sau đây chứa xếp hạng và phân tích các đặc điểm kiến trúc chung cho mẫu kiến trúc dựa trên không gian.
Xếp hạng cho từng đặc điểm dựa trên xu hướng tự nhiên đối với đặc điểm đó như một khả năng dựa trên việc triển khai mẫu điển hình, cũng như mục đích chung của mẫu. Để so sánh song song mối liên hệ của mẫu này với các mẫu khác trong báo cáo này, vui lòng tham khảo Phụ lục A ở cuối báo cáo này.

Đánh giá	mức độ
linh hoạt tổng thể	: Phân tích cao: Tính linh hoạt tổng thể là khả năng phản ứng nhanh chóng với môi trường thay đổi liên tục. Vì các đơn vị xử lý (các phiên bản đã triển khai của ứng dụng) có thể được đưa lên và xuống nhanh chóng nên ứng dụng phản ứng tốt với các thay đổi liên quan đến việc tăng hoặc giảm tải của người dùng (thay đổi môi trường). Kiến trúc được tạo bằng mẫu này thường đáp ứng tốt với các thay đổi mã hóa do kích thước ứng dụng nhỏ và tính chất động của mẫu.

Dễ triển khai Xếp hạng : Phân tích cao: Mặc dù các kiến trúc dựa trên không gian thường không được tách rời và phân tán, nhưng chúng rất năng động và các công cụ dựa trên đám mây tinh vi cho phép các ứng dụng dễ dàng được “đẩy” ra máy chủ, đơn giản hóa việc triển khai.

Xếp hạng khả năng kiểm thử : Phân tích thấp: Việc đạt được lượng người dùng rất cao trong môi trường thử nghiệm vừa tốn kém vừa tốn thời gian, gây khó khăn cho việc kiểm tra các khía cạnh khả năng mở rộng của ứng dụng.

Xếp hạng hiệu suất : Phân tích cao: Đạt được hiệu suất cao thông qua cơ chế truy cập dữ liệu trong bộ nhớ và bộ nhớ đệm được tích hợp trong mẫu này.

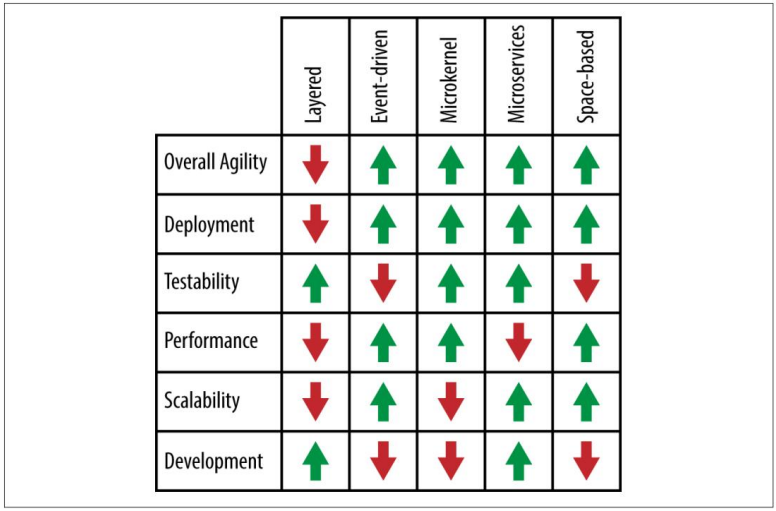
Xếp hạng khả năng mở rộng : Phân tích cao: Khả năng mở rộng cao xuất phát từ thực tế là có rất ít hoặc không có sự phụ thuộc vào cơ sở dữ liệu tập trung, do đó về cơ bản loại bỏ nút thắt hạn chế này khỏi phương trình khả năng mở rộng.

Dễ phát triển Xếp hạng: Phân tích thấp: Các sản phẩm lưới dữ liệu trong bộ nhớ và bộ nhớ đệm phức tạp làm cho mẫu này tương đối phức tạp để phát triển, chủ yếu là do thiếu sự quen thuộc với các công cụ và sản phẩm được sử dụng để tạo ra loại kiến trúc này. Hơn nữa, phải đặc biệt cẩn thận khi phát triển các loại kiến trúc này để đảm bảo không có gì trong mã nguồn ảnh hưởng đến hiệu suất và khả năng mở rộng.

PHỤ LỤC A

Tóm tắt phân tích mẫu

Hình A-1 tóm tắt cách tính điểm phân tích mẫu cho từng mẫu kiến trúc được mô tả trong báo cáo này. Bản tóm tắt này sẽ giúp bạn xác định mẫu nào có thể phù hợp nhất với tình huống của bạn.
Ví dụ: nếu mối quan tâm kiến trúc chính của bạn là khả năng mở rộng, bạn có thể xem qua biểu đồ này và thấy rằng mẫu hướng sự kiện, mẫu vi dịch vụ và mẫu dựa trên không gian có thể là những lựa chọn mẫu kiến trúc tốt. Tương tự, nếu chọn mẫu kiến trúc phân lớp cho ứng dụng của mình, bạn có thể tham khảo biểu đồ để biết rằng việc triển khai, hiệu suất và khả năng mở rộng có thể là các lĩnh vực rủi ro trong kiến trúc của bạn.



Hình A-1. Tóm tắt phân tích mẫu

Mặc dù biểu đồ này sẽ giúp hướng dẫn bạn chọn mẫu phù hợp nhưng vẫn còn nhiều điều cần cân nhắc khi chọn mẫu kiến trúc. Bạn phải phân tích tất cả các khía cạnh của môi trường của mình, bao gồm hỗ trợ cơ sở hạ tầng, bộ kỹ năng của nhà phát triển, ngân sách dự án, thời hạn dự án và quy mô ứng dụng (có thể kể tên một số khía cạnh). Việc chọn đúng mẫu kiến trúc là rất quan trọng, bởi vì một khi đã có kiến trúc thì rất khó (và tốn kém) để thay đổi.

Về tác giả

Mark Richards là một kiến trúc sư phần mềm thực hành, giàu kinh nghiệm, tham gia vào kiến trúc, thiết kế và triển khai các kiến trúc dịch vụ vi mô, kiến trúc hướng dịch vụ và các hệ thống phân tán trong J2EE và các công nghệ khác. Ông làm việc trong ngành phần mềm từ năm 1983 và có nhiều kinh nghiệm cũng như kiến thức chuyên môn về ứng dụng, tích hợp và kiến trúc doanh nghiệp.

Mark từng là chủ tịch của Nhóm người dùng Java New England từ năm 1999 đến năm 2003. Ông là tác giả của nhiều cuốn sách và video kỹ thuật, bao gồm cả Nguyên tắc cơ bản về Kiến trúc phần mềm. (video O'Reilly), Tin nhắn doanh nghiệp (Video O'Reilly), Dịch vụ tin nhắn Java , Phiên bản thứ 2 (O'Reilly) và là tác giả đóng góp cho 97 điều mọi kiến trúc sư phần mềm nên biết (O'Reilly).

Mark có bằng thạc sĩ về khoa học máy tính và có nhiều chứng chỉ về kiến trúc cũng như nhà phát triển từ IBM, Sun, The Open Group và BEA. Ông là diễn giả hội nghị thường xuyên tại Chuỗi Hội nghị chuyên đề No Fluff Just Stuff (NFJS) và đã phát biểu tại hơn 100 hội nghị và nhóm người dùng trên khắp thế giới về nhiều chủ đề kỹ thuật liên quan đến doanh nghiệp. Khi không làm việc, người ta thường thấy Mark đi bộ đường dài trên dãy núi White Mountains hoặc dọc theo Đường mòn Appalachian.