

(19)日本国特許庁(JP)

(12)公表特許公報(A)

(11)公表番号
特表2025-518555
(P2025-518555A)

(43)公表日 令和7年6月17日(2025.6.17)

(51)国際特許分類		F I	
H 0 4 L	67/60 (2022.01)	H 0 4 L	67/60
H 0 4 L	67/02 (2022.01)	H 0 4 L	67/02
H 0 4 L	67/565 (2022.01)	H 0 4 L	67/565
H 0 4 L	67/566 (2022.01)	H 0 4 L	67/566
審査請求 未請求 予備審査請求 未請求 (全68頁)			
(21)出願番号	特願2024-568603(P2024-568603)	(71)出願人	502303739
(86)(22)出願日	令和5年5月18日(2023.5.18)		オラクル・インターナショナル・コーポ レイション
(85)翻訳文提出日	令和6年12月19日(2024.12.19)		アメリカ合衆国 9 4 0 6 5 カリフォル ニア州 レッドウッド ショアーズ、メー ル ストップ 5 オービー 7 オラクル パ ークウェイ 5 0 0
(86)国際出願番号	PCT/US2023/022765	(74)代理人	110001195
(87)国際公開番号	WO2023/225219		弁理士法人深見特許事務所
(87)国際公開日	令和5年11月23日(2023.11.23)	(72)発明者	ランガー, トマス
(31)優先権主張番号	63/343,934		アメリカ合衆国、9 4 0 6 5 カリフォ ルニア州、レッドウッド・ショアーズ、 メール・ストップ・5・オー・ビー・7 、オラクル・パークウェイ、5 0 0
(32)優先日	令和4年5月19日(2022.5.19)		
(33)優先権主張国・地域又は機関	米国(US)		
(31)優先権主張番号	63/392,039		
(32)優先日	令和4年7月25日(2022.7.25)		
(33)優先権主張国・地域又は機関	米国(US)		
(31)優先権主張番号	18/198,759		
最終頁に続く			

(54)【発明の名称】 サーバコンピューティング環境におけるヘッダ処理のためのシステムと方法

(57)【要約】

一実施形態によれば、本明細書では、クラウドネイティブなアプリケーションまたはサービスをビルドするために使用可能な関連するライブラリおよび機能とともにウェブサーバを含む、マイクロサービスまたは他のコンピューティング環境で使用されるシステムおよび方法について説明する。さまざまな実施形態によれば、システムおよび方法は、例えば、(a)ヘッダ処理、(b)クライアントおよびサーバ接続の抽象化、(c)ルータの抽象化、および(d)接続のプロトコルの識別をサポートする、さまざまなコンポーネントまたは機能の使用を含むことができる。

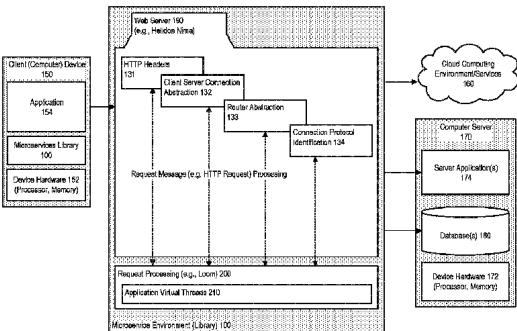


FIGURE 8

【特許請求の範囲】

【請求項 1】

マイクロサービスまたは他のコンピューティング環境で使用されるシステムであって、
1つまたは複数のプロセッサを含み、ソフトウェアアプリケーションまたはサービスとともに使用されるマイクロサービスまたは他のコンピューティング環境へのアクセスを提供するコンピュータ、
を備え、

前記システムは、要求メッセージに関連付けられたネットワーク通信プロトコルの識別を可能にするソフトウェアメソッドを提供し、

前記ネットワーク通信プロトコルの前記識別は、ヘッダ処理、クライアントーサーバ接続の抽象化、ルータの抽象化、またはソケット上での複数のネットワーク通信プロトコルプロトコルの使用に対するサポートのうちの1つまたは複数を提供するために使用される、システム。

10

【請求項 2】

前記システムは、ヘッダ列挙を介して、メッセージヘッダがオブジェクトとして扱われることを可能にする抽象化を提供し、前記ヘッダ列挙は、複数の通信プロトコルをサポートし、クライアントおよびサーバが、いくつかのサポートされるプロトコルのうちの任意のものを使用して要求メッセージおよび応答メッセージを通信することを可能にするアプリケーションプログラムインターフェースを介してアクセス可能である、請求項 1 に記載のシステム。

20

【請求項 3】

既知のヘッダタイプを有する要求メッセージに遭遇したとき、前記システムは、利用可能であれば、前記列挙からインデックス付きの値を取得し、関連付けられたデータを、前記要求が向けられている前記プロセスに直接提供し、および／または、後の使用のために前記ヘッダの値をキャッシュすることができる、請求項 2 に記載のシステム。

【請求項 4】

ソフトウェアアプリケーションまたはサービスのテスト中に、クライアント接続が、ネットワーク通信プロトコルの使用をエミュレートし、サーバ接続に逆関数を公開するインメモリ実装に置き換えられ、

前記クライアント接続のデータライタに書き込まれたデータが、前記サーバ接続においてデータリーダを通して利用可能になること、および、

30

前記サーバ接続のデータライタに書き込まれたデータが、前記クライアント接続においてデータリーダを通して利用可能になること
を含む、請求項 1 に記載のシステム。

【請求項 5】

前記インメモリ実装は、メモリ内のブロッキングキューを使用し、実際のソケットを使用せずに前記ネットワーク通信プロトコルの使用をエミュレートし、前記テストは完了しており、前記クライアントおよび／またはサーバコードは、実際のソケットに対して実行することができる、請求項 4 に記載のシステム。

【請求項 6】

40

前記システムは、複数のネットワークプロトコルまたは通信プロトコルを介して要求に対応するように動作することができるサーバの構成を可能にし、

前記サーバによってサポートされる各ネットワークプロトコルまたは通信プロトコルを、前記ネットワークプロトコルまたは通信プロトコル自体のルーティングと関連付けるルート抽象化が提供されること、および、

前記ルート抽象化は、特定のネットワークプロトコルまたは通信プロトコルで使用される1つまたは複数の特定のルートを含むことができること
を含む、請求項 1 に記載のシステム。

【請求項 7】

特定のプロトコルを介して受信された要求メッセージに対して、前記システムは、前記

50

要求に対する適切なルートを選択し、前記ルート抽象化に従って前記要求を処理することができる、請求項 6 に記載のシステム。

【請求項 8】

要求メッセージおよびクライアントーサーバ接続に関連付けられたネットワーク通信プロトコルの識別を可能にするソフトウェアメソッドを提供する拡張可能なサービスプロバイダインターフェースをさらに備え、

前記ネットワーク通信プロトコルの前記識別は、クライアントーサーバ接続の抽象化、ルータの抽象化、または単一のソケット上での複数のネットワーク通信プロトコルプロトコルの前記使用に対するサポートのうちの 1 つまたは複数を提供するために使用される、請求項 1 に記載のシステム。

10

【請求項 9】

接続要求メッセージがクライアントから受信され、サーバに向けられたとき、前記要求を、提供されたネットワーク通信プロトコルを使用して処理することができるか否かに関して、判定が行われ、プロトコル識別プロセスが、前記システムが前記要求ネットワーク通信プロトコルをサポートしていないこと、または前記システムが現在、判断を行うのに十分なデータを有していないこと、またはそのような判定を行うために追加のデータが必要であることの標示を返す、請求項 8 に記載のシステム。

【請求項 10】

前記システムは、ウェブサーバの一部として、またはウェブサーバとともに使用されるように提供され、前記要求は HTTP 要求である、請求項 1 に記載のシステム。

20

【請求項 11】

前記システムは、クラウドネイティブサービスをビルドするために使用可能な関連するライブラリおよび機能とともに仮想スレッドを使用する要求処理コンポーネントを含む、請求項 1 に記載のシステム。

【請求項 12】

前記システムは、1 つまたは複数のクラウド、データベース、または他のシステムもしくはサービスへのアクセスを提供するクラウドコンピューティング環境内で提供される、請求項 1 に記載のシステム。

【請求項 13】

マイクロサービスまたは他のコンピューティング環境で使用される方法であって、

ソフトウェアアプリケーションまたはサービスとともに使用されるマイクロサービスまたは他のコンピューティング環境を提供することと、

要求メッセージに関連付けられたネットワーク通信プロトコルの識別を可能にするソフトウェアメソッドを提供することと、
を含み、

30

前記ネットワーク通信プロトコルの前記識別は、ヘッダ処理、クライアントーサーバ接続の抽象化、ルータの抽象化、またはソケット上での複数のネットワーク通信プロトコルプロトコルの使用に対するサポートのうちの 1 つまたは複数を提供するために使用される、方法。

【請求項 14】

システムが、ヘッダ列挙を介して、メッセージヘッダがオブジェクトとして扱われることを可能にする抽象化を提供することをさらに含み、前記ヘッダ列挙は、複数の通信プロトコルをサポートし、クライアントおよびサーバが、いくつかのサポートされるプロトコルのうちの任意のものを使用して要求メッセージおよび応答メッセージを通信することを可能にするアプリケーションプログラムインターフェースを介してアクセス可能である、請求項 13 に記載の方法。

40

【請求項 15】

既知のヘッダタイプを有する要求メッセージに遭遇したとき、前記システムは、利用可能であれば、前記列挙からインデックス付きの値を取得し、関連付けられたデータを、前記要求が向けられている前記プロセスに直接提供し、および／または、後の使用のために

50

前記ヘッダの値をキャッシュすることができる、請求項 14 に記載の方法。

【請求項 16】

ソフトウェアアプリケーションまたはサービスのテスト中に、クライアント接続を、ネットワーク通信プロトコルの使用をエミュレートし、サーバ接続に逆関数を公開するインメモリ実装に置き換えることをさらに含み、

前記クライアント接続のデータライタに書き込まれたデータが、前記サーバ接続においてデータリーダを通して利用可能になること、および、

前記サーバ接続のデータライタに書き込まれたデータが、前記クライアント接続においてデータリーダを通して利用可能になること

を含む、請求項 13 に記載の方法。

10

【請求項 17】

前記インメモリ実装は、メモリ内のブロッキングキューを使用し、実際のソケットを使用せずに前記ネットワーク通信プロトコルの使用をエミュレートし、前記テストは完了しており、前記クライアントおよび／またはサーバコードは、実際のソケットに対して実行することができる、請求項 16 に記載の方法。

【請求項 18】

複数のネットワークプロトコルまたは通信プロトコルを介して要求に対応するように動作することができるサーバの構成を可能にするシステムによって実行され、

前記サーバによってサポートされる各ネットワークプロトコルまたは通信プロトコルを、前記ネットワークプロトコルまたは通信プロトコル自体のルーティングと関連付けるルート抽象化が提供されること、および、

前記ルート抽象化は、特定のネットワークプロトコルまたは通信プロトコルで使用される 1 つまたは複数の特定のルートを含むことができること

を含む、請求項 13 に記載の方法。

20

【請求項 19】

特定のプロトコルを介して受信された要求メッセージに対して、前記システムは、前記要求に対する適切なルートを選択し、前記ルート抽象化に従って前記要求を処理することができる、請求項 18 に記載の方法。

【請求項 20】

要求メッセージおよびクライアントーサーバ接続に関連付けられたネットワーク通信プロトコルの識別を可能にするソフトウェアメソッドを提供する拡張可能なサービスプロバイダインターフェースを提供することを含み、

前記ネットワーク通信プロトコルの前記識別は、クライアントーサーバ接続の抽象化、ルータの抽象化、および単一のソケット上での複数のネットワーク通信プロトコルプロトコルの前記使用のうちの 1 つまたは複数を提供するために使用される、請求項 13 に記載の方法。

30

【請求項 21】

接続要求メッセージがクライアントから受信され、サーバに向けられたとき、前記要求を、提供されたネットワーク通信プロトコルを使用して処理することができるか否かに関して、判定が行われ、プロトコル識別プロセスが、前記システムが前記要求ネットワーク通信プロトコルをサポートしていないこと、または前記システムが現在、判断を行うのに十分なデータを有していないこと、またはそのような判定を行うために追加のデータが必要であることの標示を返す、システムによって実行される請求項 20 に記載の方法。

40

【請求項 22】

前記方法は、ウェブサーバの一部として、またはウェブサーバとともに使用されるように提供され、前記要求は HTTP 要求である、請求項 13 に記載の方法。

【請求項 23】

前記方法は、クラウドネイティブサービスをビルドするために使用可能な関連するライブラリおよび機能とともに仮想スレッドを使用する要求処理コンポーネントで動作する、請求項 13 に記載の方法。

50

【請求項 24】

前記方法は、1つまたは複数のクラウド、データベース、または他のシステムもしくはサービスへのアクセスを提供するクラウドコンピューティング環境内で動作する、請求項13に記載の方法。

【請求項 25】

1つまたは複数のコンピュータによって読み出され実行されると、前記1つまたは複数のコンピュータに、

ソフトウェアアプリケーションまたはサービスとともに使用されるマイクロサービスまたは他のコンピューティング環境を提供することと、

要求メッセージに関連付けられたネットワーク通信プロトコルの識別を可能にするソフトウェアメソッドを提供することと、

を含み、

前記ネットワーク通信プロトコルの前記識別は、ヘッダ処理、クライアント-サーバ接続の抽象化、ルータの抽象化、またはソケット上での複数のネットワーク通信プロトコルプロトコルの使用に対するサポートのうちの1つまたは複数を提供するために使用される、

方法を実行させる命令が格納されている、非一時的コンピュータ可読記憶媒体。

【発明の詳細な説明】

【技術分野】

【0001】

著作権表示

本特許文献の開示の一部には、著作権保護の対象となる資料が含まれている。著作権者は、特許商標庁の特許ファイルまたは記録に記載されているため、本特許文献または本特許開示の何人による複製に対しても何ら異議を唱えないが、他の場合には、いかなる場合もすべての著作権を留保する。

【0002】

優先権の主張

本出願は、2022年5月19日に出願された出願番号第63／343，934号である「SYSTEMS AND METHODS FOR USE WITH A MICROSERVICES OR OTHER COMPUTING ENVIRONMENT」と題する米国仮特許出願、2022年7月25日に出願された出願番号第63／392，039号である「SYSTEMS AND METHODS FOR USE WITH A MICROSERVICES OR OTHER COMPUTING ENVIRONMENT」と題する米国仮特許出願、2023年5月17日に出願された出願番号第18／198，759号である「SYSTEMS AND METHODS FOR HEADER PROCESSING IN A SERVER COMPUTING ENVIRONMENT」と題する米国特許出願、2023年5月17日に出願された出願番号第18／198，762号である「SYSTEMS AND METHODS FOR CLIENT-SERVER CONNECTION ABSTRACTION IN A SERVER COMPUTING ENVIRONMENT」と題する米国特許出願、2023年5月17日に出願された出願番号第18／198，764号である「SYSTEMS AND METHODS FOR ROUTER ABSTRACTION IN A SERVER COMPUTING ENVIRONMENT」と題する米国特許出願、および2023年5月17日に出願された出願番号第18／198，765号である「SYSTEMS AND METHODS FOR IDENTIFYING CONNECTION PROTOCOLS IN A SERVER COMPUTING ENVIRONMENT」と題する米国特許出願に対する優先権の利益を主張し、これらの出願の各々およびそれらの内容は、引用により本明細書に援用される。

【0003】

技術分野

本明細書で説明する実施形態は、概して、クラウドコンピューティングおよび他のコンピューティング環境、ソフトウェア開発、ならびにマイクロサービスアーキテクチャに関し、特に、クラウドネイティブなアプリケーションまたはサービスをビルドするために使

用可能な関連するライブラリおよび機能とともに、アプリケーション開発をサポートするためのシステムおよび方法を対象とする。

【背景技術】

【0004】

背景

マイクロサービス環境は、ソフトウェアアプリケーションを、独立してデプロイ可能であり、ネットワーク、例えばクラウドコンピューティング環境によって提供される可能性のあるようなクラウドネットワークを介して互いに通信する、疎結合サービスの集まりとして提示する。マイクロサービスアプローチは、例えば、クラウドネイティブアプリケーションまたはサービスとしてクラウドコンピューティング環境で提供すべきソフトウェアアプリケーションまたはサービスを開発するために使用することができる。

10

【発明の概要】

【0005】

概要

一実施形態によれば、本明細書では、クラウドネイティブなアプリケーションまたはサービスをビルドするために使用可能な関連するライブラリおよび機能とともにウェブサーバを含む、マイクロサービスまたは他のコンピューティング環境で使用されるシステムおよび方法について説明する。

【0006】

さまざまな実施形態によれば、システムおよび方法は、例えば、(a) HTTP ヘッダ、(b) クライアントおよびサーバ接続の抽象化、(c) ルータの抽象化、および (d) 接続のプロトコルの識別をサポートする、さまざまなコンポーネントまたは機能の使用を含むことができる。

20

【0007】

一実施形態によれば、システムは、ヘッダ列挙によって、メッセージヘッダが、アプリケーションプログラムインターフェース (API) を介してアクセス可能な、オブジェクトとして扱われることを可能にする抽象化を提供し、API は、複数の通信プロトコルをサポートし、クライアントおよびサーバが、サポートされているプロトコルのうちの任意のものを使用して要求／応答メッセージを通信することを可能にする。既知のヘッダタイプを有する要求メッセージに遭遇したとき、システムは、利用可能であれば、列挙からインデックス付きの値を取得し、関連付けられたデータを (例えばデータバイトとして) 要求が向けられているプロセスに直接提供し、および／または後の使用のためにヘッダの値をキャッシュすることができる。

30

【0008】

一実施形態によれば、例えば、ソフトウェアアプリケーションまたはサービスのテストをサポートするために、クライアント接続は、テストの目的で、例えば HTTP を呼び出さないが、代わりにサーバ接続に逆関数を公開するインメモリ実装に置き換えることができ、例えば、クライアント接続のデータライタに書き込まれたデータバイトは、サーバ接続においてデータリーダを通して利用可能であり、その逆も同様である。テスト中、インメモリ実装は、メモリ内のブロッキングキューを使用し、実際のソケットを使用せずに HTTP プロトコルの使用をエミュレートする。インメモリ実装を使用するテスト (例えば、単体テストまたは統合テスト) が完了すると、要求を処理するのに使用するために、クライアントコードおよび／またはサーバコードを実際のソケットに対してテストすることができる。

40

【0009】

一実施形態によれば、システムは、例えば HTTP 1、gRPC over HTTP 2、および／または WebSocket などの複数のネットワークまたは通信プロトコルを介して要求に対応するように動作することができるサーバの構成を可能にする。ルートの抽象化により、各プロトコルが、それ自体のルーティング、例えば HTTP ルーティングを有することができる。下位抽象化 (sub-abstraction) が、例えば HTTP / 1、

50

1 ルートまたは H T T P / 2 ルートなど、より具体的なルートを提供することができる。特定のプロトコルを介して受け取った要求メッセージに対して、システムは、その要求に対して適切なルートを選択し、それに従って要求を処理することができる。

【 0 0 1 0 】

一実施形態によれば、クライアント／サーバ接続の抽象化、ルータの抽象化、または単一のソケット上での異なるプロトコルの使用などの機能をサポートするために、拡張可能なサービスプロバイダインタフェース（ S P I ）が、要求メッセージおよび接続のためのプロトコル識別を可能にするソフトウェアメソッドを提供する。クライアントからサーバに向けられた接続要求メッセージが受信されると、その要求を適切に処理することができるか否かに関する判定が行われる。プロトコル識別プロセスは、例えば、システムが要求プロトコルをサポートしていないこと、システムが判断を下すのに十分なデータを有していないこと、またはそのような判定を行うためには追加のデータが必要であることを示す標示を返すことができる。

【 図面の簡単な説明 】

【 0 0 1 1 】

【 図 1 】 一実施形態による、ソフトウェア開発フレームワークを提供するマイクロサービス環境のさまざまな例を示す図である。

【 図 2 】 一実施形態による、ソフトウェア開発フレームワークを提供するマイクロサービス環境のさまざまな例をさらに示す図である。

【 図 3 】 一実施形態による、ソフトウェア開発フレームワークを提供するマイクロサービス環境のさまざまな例をさらに示す図である。

【 図 4 】 一実施形態による、ソフトウェア開発フレームワークを提供するマイクロサービス環境のさまざまな例をさらに示す図である。

【 図 5 】 一実施形態による、クラウド環境で使用されるアプリケーションを開発するために使用されるようなマイクロサービス環境を含むシステムを示す図である。

【 図 6 】 一実施形態による、クラウド環境で使用されるアプリケーションを開発するためのマイクロサービス環境の使用をさらに示す図である。

【 図 7 】 一実施形態による、マイクロサービス環境を含むシステムを示す図である。

【 図 8 】 一実施形態による、マイクロサービス環境で使用されるウェブサーバを示す図である。

【 図 9 】 一実施形態による、 H T T P ヘッダ処理に対するサポートを示す図である。

【 図 1 0 】 一実施形態による、 H T T P ヘッダ処理に対するサポートをさらに示す図である。

【 図 1 1 】 一実施形態による、 H T T P ヘッダ処理に対するサポートをさらに示す図である。

【 図 1 2 】 一実施形態による、 H T T P ヘッダ処理に対するサポートをさらに示す図である。

【 図 1 3 】 一実施形態による、 H T T P ヘッダを処理するための、マイクロサービス環境で使用される方法を示す図である。

【 図 1 4 】 一実施形態による、クライアントおよびサーバの抽象化に対するサポートを示す図である。

【 図 1 5 】 一実施形態による、クライアントおよびサーバの抽象化に対するサポートをさらに示す図である。

【 図 1 6 】 一実施形態による、クライアントおよびサーバの抽象化に対するサポートをさらに示す図である。

【 図 1 7 】 一実施形態による、クライアントおよびサーバの抽象化に対するサポートをさらに示す図である。

【 図 1 8 】 一実施形態による、クライアントおよびサーバの抽象化を提供するための、マイクロサービス環境で使用される方法を示す図である。

【 図 1 9 】 一実施形態による、ルータの抽象化に対するサポートを示す図である。

10

20

30

40

50

【図 20】一実施形態による、ルータの抽象化に対するサポートをさらに示す図である。
【図 21】一実施形態による、ルータの抽象化に対するサポートをさらに示す図である。
【図 22】一実施形態による、ルータの抽象化に対するサポートをさらに示す図である。
【図 23】一実施形態による、ルータの抽象化を提供するための、マイクロサービス環境で使用される方法を示す図である。

【図 24】一実施形態による、接続のプロトコルを識別するためのサポートを示す図である。

【図 25】一実施形態による、接続のプロトコルを識別するためのサポートをさらに示す図である。

【図 26】一実施形態による、接続のプロトコルを識別するためのサポートをさらに示す図である。

【図 27】一実施形態による、接続のプロトコルを識別するためのサポートをさらに示す図である。

【図 28】一実施形態による、接続のプロトコルを識別するための、マイクロサービス環境で使用される方法を示す図である。

【発明を実施するための形態】

【0012】

詳細な説明

マイクロサービス環境は、ソフトウェアアプリケーションを、独立してデプロイ可能であり、ネットワーク、例えばクラウドコンピューティング環境（クラウド環境）によって提供される可能性のあるようなクラウドネットワークを介して互いに通信する、疎結合サービスの集まりとして提示する。マイクロサービスアプローチは、例えば、クラウドネイティブアプリケーションまたはサービスとしてクラウドコンピューティング環境で提供すべきソフトウェアアプリケーションまたはサービスを開発するために使用することができる。

【0013】

マイクロサービスの開発において、Helidonなどのソフトウェア開発フレームワークを使用することができる。例えば、後述するように、Helidon環境は、構成、セキュリティ、またはウェブサーバ機能性などの機能をサポートし、ソフトウェアアプリケーション開発者に、クラウドネイティブなアプリケーションまたはサービスを開発するための基礎を提供する、ソフトウェアライブラリのコレクションを提供する。

【0014】

一実施形態によれば、Helidonライブラリは、例えばDocker、Kubernetes、Prometheus、またはJaegerなど、他のソフトウェア開発、デプロイメント、および／またはモニタリングツールと相互運用することができる。

【0015】

マイクロサービス環境

図1～図4は、一実施形態による、ソフトウェア開発フレームワークを提供するマイクロサービス環境のさまざまな例を示す。

【0016】

一実施形態によれば、図1～図4に示し、さまざまな実施形態に関して本明細書でさらに説明するコンポーネントおよびプロセスは、コンピュータシステムまたは他のタイプの処理デバイス、例えばクラウドコンピューティングシステムによって実行可能なコンピュータソフトウェアまたはプログラムコードとして提供することができる。

【0017】

図1に示すように、一実施形態によれば、（例えば、マイクロサービスライブラリによって提供されるような）マイクロサービス環境100は、例えば、1つまたは複数のHelidon SE110、Helidon MP120、および／またはHelidon Nima 130プログラミングモデルまたは環境を提供することができる。マイクロサービス環境は、開発されたアプリケーションによるクラウド、データベース、または他の

10

20

30

40

50

システムもしくはサービス 140 との相互作用を可能にすることができる。

【0018】

本明細書で説明する Helidon マイクロサービス環境のさまざまな実施形態の例は、マイクロサービスコンピューティング環境またはマイクロサービス環境の例を説明する目的で提供する。さまざまな実施形態によれば、本明細書に記載するシステムおよび方法は、他のタイプのマイクロサービス環境またはコンピューティング環境で 사용할 ことができる。

【0019】

さまざまな実施形態によれば、システムおよび方法は、他のタイプのソフトウェア開発環境で、またはクラウドネイティブサービスをビルドするために使用可能な関連するライブラリおよび機能とともに他のタイプのウェブサーバで、使用することができる。

【0020】

図 2 に示すように、一実施形態によれば、Helidon SE 環境は、ウェブサーバ、セキュリティ、および構成コンポーネントを直接使用する関数型プログラミングスタイルをサポートし、アプリケーション開発者に透明性および制御を提供し、リアクティブストリーム、および非同期の関数型プログラミングなどの Java 機能をサポートする。

【0021】

例えば、一実施形態によれば、Helidon SE 環境は、アプリケーション開発者が軽量のリアクティブマイクロサービスをビルドすることができるフレームワークを提供する。この環境は、例えば、ウェブアプリケーションを作成するための非同期でリアクティブな API を提供するリアクティブウェブサーバ 111、キー／値形式の構成プロパティをロードおよび処理して、後でアプリケーションがコンフィグデータを取り出すために使用することができるコンフィグオブジェクトにするための Java API を提供する構成 API 112、ならびに、認証、認可、およびアウトバウンドセキュリティを提供するセキュリティコンポーネント 113 など、さまざまなライブラリ、アプリケーションプログラムインターフェース (API)、または他のコンポーネントを含むことができ、また、メトリクス 114、ヘルスチェック 115、およびトレース 116 または他のコンポーネントを含むこともできる。

【0022】

図 3 に示すように、一実施形態によれば、Helidon MP 環境は、Helidon ライブラリの上にビルドされた API の MicroProfile ファミリーを使用することによって、宣言型プログラミングスタイルをサポートする。(例えば、Eclipse MicroProfile プロジェクトによって指定されるような) MicroProfile 定義を使用して、複数の MicroProfile ランタイムにわたってアプリケーションポータビリティをサポートすることができる。

【0023】

例えば、一実施形態によれば、Helidon MP 環境は、例えば、JAX-RS 122、JSON-P 126、コンテキストおよび依存関係の注入 (CDI: Contexts and Dependency Injection) 124、メトリクス 121、ヘルスチェック 123、フォールトトレランス 125、MicroProfile (MP) 構成 137、および JSON ウェブトークン (JWT) 認証 128 コンポーネントなど、さまざまなライブラリ、API、または他のコンポーネントを含むことができる。ウェブサーバは、例えば Netty などのノンブロッキングなクライアント／サーバ／ウェブフレームワーク 118 によって提供することができる。

【0024】

図 4 に示すように、一実施形態によれば、Helidon Nima 環境は、クラウドネイティブサービスをビルドするために使用可能な関連するライブラリおよび機能とともに、ウェブサーバの Loom ベースの実装を提供する。

【0025】

例えば、一実施形態によれば、Helidon Nima 環境は、例えば、HTTP ヘ

10

20

30

40

50

ッダ 1 3 1、クライアントおよびサーバ接続抽象化 1 3 2、ルータ抽象化 1 3 3、および接続プロトコル識別 1 3 4をサポートするさまざまなコンポーネントまたは機能の使用を含むことができ、これによってアプリケーション開発者は、例えば、（リアクティブではなく）ブロッキングなマイクロサービスをビルドすることができる。

【0026】

図5は、一実施形態による、クラウド環境で使用されるアプリケーションを開発するために使用されるようなマイクロサービス環境を含むシステムを示す。

【0027】

図5に示すように、一実施形態によれば、マイクロサービス環境は、ソフトウェアアプリケーションを、独立してデプロイ可能であり、ネットワークを介して、例えばクラウドコンピューティング環境によって提供することができるようなクラウドネットワークを介して互いに通信する、疎結合サービスの集まりとして提示する。

【0028】

例えば、一実施形態によれば、Helidonマイクロサービス環境は、接続されたシステムをビルドするために、（クライアントおよび／またはサーバ）アプリケーションがマイクロサービス環境内で通信することを可能にする、リモートプロシージャコール（例えば、gRPC）フレームワークまたはコンポーネントの使用をサポートすることができる。

【0029】

一実施形態によれば、リモートプロシージャコールフレームワークは、リモートで呼び出すことができるサービスおよびメソッドの定義を有効にする。サーバまたはサービスが、クライアントアプリケーションがサーバアプリケーション上のメソッドをローカルオブジェクトであるかのように直接呼び出すことを可能にするローカルオブジェクト（スタブ）を介して、クライアントからの呼び出しを処理することができる。サーバ／サービスは、着信要求を復号する、サービスメソッドを実行する、およびサービス応答を符号化するなど、クライアント呼び出しを処理するように、メソッドを実装する。ローカルオブジェクト（スタブ）は、サービスと同じメソッドを実装し、呼び出しのためのパラメータを適切なプロトコルバッファメッセージタイプでラッピングし、これらのパラメータはその後、要求としてサーバに提供される。

【0030】

一実施形態によれば、（例えば、マイクロサービスライブラリによって提供されるような）マイクロサービス環境が、デバイスハードウェア152（例えば、152A、152B）を有するクライアントコンピュータデバイス150（例えば、150A、150B）において、クライアントアプリケーション154（例えば、154A、154B）が、マイクロサービスと通信するか、または、クラウド160、デバイスハードウェア172を有するコンピュータサーバデバイス170におけるサーバアプリケーション174、データベース180、または他のシステムもしくはサービスと、データにアクセスし、トランザクションを処理し、またはそれらのシステムもしくはサービスに関連付けられた他の動作を実行することを目的としてやり取りするように、アクセスするのを可能にする。

【0031】

図6は、一実施形態による、クラウド環境で使用されるアプリケーションを開発するためのマイクロサービス環境の使用をさらに示す。

【0032】

図6に示すように、一実施形態によれば、マイクロサービス環境が、クライアントアプリケーションが、クラウドネイティブサービス180と通信するか、または他の方法でクラウド、データベース、または他のシステムもしくはサービスと、データにアクセスし、トランザクションを処理し、またはそれらのシステムもしくはサービスに関連付けられた他の動作を実行することを目的としてやり取りするように、アクセスするのを可能にする。

【0033】

10

20

30

40

50

HTTP 要求処理

クラウド環境では、クライアントアプリケーションがサーバとやり取りすることを可能にするウェブサービスは、HTTP ベースである可能性があり、すなわち、クライアントアプリケーションおよびサーバは、データメッセージの形式でさまざまな参加者間で要求および応答を通信する手段として、HTTP 通信プロトコルを使用することができる。

【0034】

例えば、1つまたは複数のデータベースまたは他のサービスへのアクセスを提供するクラウドサービスでは、クライアント要求の処理は、いくつかの初期呼び出しになるように意図されている場合があり、そうした呼び出しの各々は、その後、要求を処理するいくつかのさらなる態様を実行するために別のサービスを求めることができる。

10

【0035】

いくつかのクラウド環境では、リアクティブプログラミング技術を使用して、要求の並行処理をサポートすることができる。しかしながら、リアクティブプログラミングはノンブロッキングであるため、リアクティブアプリケーションの動作をトラブルシューティングすることは困難である可能性がある。例えば、本来はさまざまなスレッドのステータスを評価するのに有用である可能性があるスレッドダンプは、リアクティブ環境で使用される場合、テストまたはデバッグの目的のための情報をほとんどまたはまったく提供しない。

【0036】

一実施形態によれば、このような考慮事項に対処するために、システムは、要求処理コンポーネント、例えばウェブサーバ (Web Server) の Loom ベースの実装を、(本明細書では、いくつかの実施形態では Helidon Nima と称する) クラウドネイティブサービスをビルドするために使用可能な関連するライブラリおよび機能とともに含むことができる。

20

【0037】

一実施形態によれば、このような要求処理コンポーネントは、仮想スレッドを使用してクライアント-サーバ通信の並行性をサポートする。このような仮想スレッドにより、クライアント HTTP アプリケーションは、通常のブロッキング方式で動作する、例えばサーバプロセスへの接続を要求することができ、一方で、基礎となる要求処理 (例えば、Loom ベースの) 環境は、要求処理の並行性を処理する。

30

【0038】

これにより、アプリケーション開発者がアプリケーション自体にリアクティブプログラミングを組み込む必要がなくなるという点で、コード開発プロセスが改善される。さらに、テストまたはデバッグ中に、スレッドダンプが、トラブルシューティングの目的のためにより有用な情報を提供する。このように、このアプローチにより、従来のプログラミング環境の機能とリアクティブプログラミングの並行性の利点とが組み合わせられて提供される。

【0039】

例えば、一実施形態によれば、Helidon Nima 環境によって、アプリケーション開発者が、サーバがウェブクライアントにサービスを提供することができる HTTP エンドポイントをセットアップすることができる。クライアントアプリケーションが、各接続に対して仮想スレッドを提供する仮想スレッディングを利用しながら、高レベルの並行性が達成されるように、環境内においてブロッキング方式で実行することができる。

40

【0040】

図 7 は、一実施形態による、マイクロサービス環境を含むシステムを示す。

【0041】

図 7 に示すように、一実施形態によれば、ウェブサーバ 190 (Web Server、例えば、Helidon Nima 環境) で使用される場合、要求処理コンポーネント 200、例えばウェブサーバの Loom ベースの実装は、サーバに向けられた要求がアプリケーション仮想スレッド 210 上で実行することを可能にするブロッキング API の使用

50

を含む、多数のプラットフォームスレッドが使用されるという要件なしに、ブロッキングコードが使用されることを可能にする。

【0042】

一実施形態によれば、(いくつかの例ではファイバと称する)仮想スレッドは、高スループットの並行アプリケーションの実行を可能にする軽量スレッドとして動作する。一般に説明されるように、仮想スレッドは、特定のオペレーティングシステム(O/S)スレッドに結び付けられない実行スレッド(例えば、`java.lang.Thread`)のインスタンスを提供する。アプリケーションコードが、要求の持続時間にわたって仮想スレッドで実行することができ、仮想スレッドは、さらに、プロセッサ上で動作を実行する間、O/Sスレッドを消費する。

10

【0043】

一実施形態によれば、ノンブロッキングAPIを、同期APIによってカプセル化し、対応する動作が完了するまで仮想スレッドのさらなる実行をブロックすることによって、仮想スレッド対応にすることができる。このように、各仮想スレッドは、一定期間中断し、その後、後の時点で継続することができる。

【0044】

例えば、図7に示すように、一実施形態によれば、仮想マシン220で実行されるとき、要求は、ブロッキングAPI222で受け取ることができる。複数のキャリアスレッド224が、例えばノンブロッキングAPI232でイベントおよびコールバックを登録することによって、またはO/Sスレッドプール226およびブロッキングAPI234を介して、基礎となるオペレーティングシステム230と通信する。

20

【0045】

一実施形態によれば、このような要求処理コンポーネントにより、アプリケーション開発者は、`thread-per-request`(要求ごとのスレッド)プログラミングモデルを変更することなく、ソフトウェア開発中にノンブロッキングAPIを利用することができる。また、記載したアプローチにより、複雑なエクゼキュータサービスを処理する必要なしに、システムがアウトバウンドHTTP要求またはデータベース要求などのブロッキングコードを使用することもできる。

【0046】

図8は、一実施形態による、マイクロサービス環境で使用されるウェブサーバを示す。

30

【0047】

図8に示すように、一実施形態によれば、例えば、ウェブサーバのLOOMベースの実装などの仮想スレッドを使用するウェブサーバは、リアクティブウェブサーバ(例えば、リアクティブウェブサーバ111)および/またはノンブロッキングクライアント/サーバ/ウェブフレームワーク(例えば、フレームワーク118)によって他の方法で提供される機能性の代替または置換として提供することができる。

【0048】

上述したように、要求処理コンポーネントは、仮想スレッドを使用してクライアントサーバ通信の並行性をサポートする。このような仮想スレッドにより、クライアントHTTPアプリケーションは、通常のブロッキング方式で動作する、例えばサーバプロセスへの接続を要求することができ、その一方で、基礎となる要求処理(例えばLOOMベースの)環境は要求処理の並行性を処理する。

40

【0049】

例えば、Helidon SE環境の一実施形態によれば、要求処理コンポーネントは、通常そうした環境で提供されるリアクティブウェブサーバの代替として提供することができ、リアクティブプログラミングを使用する代わりに、アプリケーション開発者は、ウェブサーバによって提供される仮想スレッドを使用して、同様のパフォーマンス態様を提供しながら、リアクティブではなくブロッキングであるアプリケーション処理をサポートすることができる。

【0050】

50

さまざまな実施形態によれば、要求処理コンポーネントは、本明細書では、Helidonの他の／新たなバージョン（例えば、Helidon Nima）で使用するウェブサーバとして、同様に提供することができる。

【0051】

例えば、一実施形態によれば、アプリケーション開発者の観点から、アプリケーション開発者は、要求処理コンポーネント、例えば、Loomベースのウェブサーバの仮想スレッド機能を直接使用するブロッキングコード、または、それ自体はリアクティブであるが、（例えば、Loomベースの）ウェブサーバが、ウェブサーバによって提供される仮想スレッド機能を使用して、アプリケーション開発者にとって透過的に処理することができるアプリケーションコードのいずれかを、自身のアプリケーションに含めることができる。

10

【0052】

ウェブサーバ／ウェブクライアント

さまざまな実施形態によれば、システムおよび方法は、例えば、（a）HTTPヘッダ、（b）クライアントおよびサーバ接続の抽象化、（c）ルータの抽象化、ならびに（d）接続のプロトコルの識別をサポートする、さまざまなコンポーネントまたは機能の使用を含むことができる。

【0053】

一実施形態によれば、ウェブサーバ（Web Server）は、HTTP／1.1プロトコルをサポートするデフォルトの実装を有する、拡張可能なモジュールとして提供することができる。ウェブサーバSPIが、例えば、プレーンソケットを使用するHTTP／2へのアップグレード、またはWeb Socketへのアップグレードなど、HTTP／1プロトコルからのアップグレード、事前知識があるHTTP／2など、バイナリデータに基づく発見を通して拡張可能な、非HTTPプロトコルを含む他のプロトコル、および／またはHTTP／1.1、HTTP／2など、トランスポートレイヤセキュリティ（TLS：Transport Layer Security）要求のためのアプリケーションレイヤプロトコルネゴシエーション（ALPN：application layer protocol negotiation）サポートをサポートする、拡張性を提供する。

20

【0054】

一実施形態によれば、拡張モジュールは、例えば、HTTP／2プロトコル実装、事前知識がある、HTTP／1からのHTTP／2 over TCP（h2c、プレーンソケット）アップグレード、h2c（TLS）（ALPNサポート）、HTTP／2の「サブプロトコル」としてのgRPCプロトコル実装、および／またはHTTP／1からのWeb Socketプロトコル実装アップグレードを含むことができる。

30

【0055】

一実施形態によれば、ウェブサーバは、バージョン固有ルートのサポートを含む、HTTP（バージョンに依存しない）ルーティングと、非HTTPプロトコルルーティングをサポートする拡張可能なルータAPIとをサポートする。拡張モジュールによっては、独自のルーティング、例えば、デフォルトのHTTPルーティング（バージョンに依存しない）もサポートするHTTP／2、すべてのメッセージタイプ（単項、クライアントストリーム、サーバストリーム、および双方向）をサポートするプロトコル固有ルートを含むgRPC、および／または、セッションを有するリスナーをサポートするWeb Socket、すなわちプロトコル固有のルートを提供する。

40

【0056】

一実施形態によれば、ウェブクライアント（Web Client）は、HTTP／1.1プロトコルをサポートするデフォルトの実装を有する、拡張可能なモジュールとして提供することができる。ウェブクライアントSPIが、基本構成を共有して使用すべき他のプロトコルをサポートする。一実施形態によれば、以下の拡張、すなわち、HTTP／1.1クライアント、HTTP／2クライアントを含めることができる。

【0057】

50

一実施形態によれば、Web Server モジュールおよび Web Client モジュールは、例えば JUnit5 により、テスト可能であるように設計されている。システムは、選択されたアノテーションに応じて、HTTP 通信またはインメモリ通信のいずれかをサポートすることができ、例えば、@ServerTest は、統合テストを作成するために、ソケットを介した HTTP 通信を提供し、および／または @RoutingTest は、ユニットテストを作成するために、インメモリ通信を提供する。

【0058】

一実施形態によれば、追加のモジュール、例えば、「サーキットブレーカー (circuit breaker)」、「バルクヘッド (bulkhead)」、「フォールバック (fallback)」、「リトライ (retry)」シナリオのためのフォールトトレランス実装、ブロックングフォールトトレランス実装、ヘルス、メトリクス、構成、アプリケーション情報など、アプリケーションに関する情報を取得するための可観測性エンドポイント、「オブザーブ (observe)」メカニズムに他のモジュールをプラグインするための拡張性を提供する手段、および／またはトレース、もしくはオープンテレメトリのトレースサポートを含めることができる。

【0059】

A. HTTP ヘッダ

異なる通信プロトコルを介してクライアントーサーバ通信をサポートするクラウド環境内では、各通信プロトコルが、メッセージヘッダを含む、参加者間で渡されるメッセージのフォーマット化に関する独自の要件を有する場合がある。

【0060】

例えば、HTTP / 1 プロトコルと HTTP / 2 プロトコルとは、それらのプロトコルを介して要求メッセージまたは応答メッセージを送信するときに使用されるメッセージヘッダのフォーマットに関して、異なる要件を有する。

【0061】

このようなヘッダをパースするには、特にクラウド環境で遭遇する可能性があるように、何千もの要求を同時に処理する必要がある場合には、かなりの量の計算オーバーヘッドが必要になる。このような要求に対する応答をそれらの適切なヘッダで準備することも、同様にかなりの量の計算オーバーヘッドを必要とする。

【0062】

一実施形態によれば、システムは、ヘッダ列挙によって、メッセージヘッダが、アプリケーションプログラムインターフェース (API) を介してアクセス可能な、オブジェクトとして扱われることを可能にする抽象化を提供し、API は、複数の通信プロトコルをサポートし、クライアントおよびサーバが、サポートされているプロトコルのうちの任意のものを使用して要求／応答メッセージを通信することを可能にする。

【0063】

既知のヘッダタイプを有する要求メッセージに遭遇したとき、システムは、利用可能であれば、列挙からインデックス付きの値を取得し、関連付けられたデータを（例えばデータバイトとして）要求が向けられているプロセスに直接提供し、および／または後の使用のためにヘッダの値をキャッシュすることができる。

【0064】

例えば、HTTP ヘッダのパース、処理、およびシリアライゼーションは、HTTP 要求の処理における、計算コストのより高い部分の 1 つである。これらのタスクのすべてにおけるパフォーマンスを向上させるために、システムは、ヘッダの名前 (インターフェース HeaderName) とヘッダの値全体 (インターフェース HeaderValue) の両方について、ヘッダの新たな抽象化を提供することができる。

【0065】

さらに、HTTP / 2 がますます頻繁に使用されるようになってきているため、両方の HTTP バージョンでうまく機能するヘッダを使用することは有利である。Java での実装は、ヘッダを識別するために String または CharSequence を使用するか、またはヘッダ

10

20

30

40

50

名のバイト値を最適化する可能性があるが、一般的に使用されるヘッダに対してインデックス付けされたアプローチを使用しない。

【0066】

説明するアプローチは、各要求に対してヘッダマップをルックアップする必要性を回避し、特にクラウド環境で遭遇する可能性があるような大量の要求を処理するときに、計算の節約を可能にする。

【0067】

図9は、一実施形態によるHTTPヘッダ処理に対するサポートを示す。

【0068】

図9に示すように、一実施形態によれば、システムは、ヘッダ列挙250によって、メッセージヘッダが、アプリケーションプログラムインターフェース（API）を介してアクセス可能な、オブジェクトとして扱われることを可能にする抽象化を提供し、APIは、複数の通信プロトコルをサポートするし、クライアントおよびサーバが、サポートされているプロトコルのうちの任意のものを使用して要求／応答メッセージを通信することを可能にする。

【0069】

一実施形態によれば、既知のヘッダタイプ（例えば、A1 252、A2 253、B1 256、B2 257）を有する要求メッセージに遭遇したとき、システムは、利用可能であれば、列挙からインデックス付きの値（例えば、A254、B258）を取得し、関連付けられたデータを（例えば、データバイトとして）要求が向けられているプロセスに直接提供し、および／または後の使用のためにヘッダ値をキャッシュする（270）ことができる。

【0070】

一実施形態によれば、HTTPヘッダはオブジェクト型として使用される。既知のヘッダが作成されると、システムは、値のデータバイト（単一値の場合）、ヘッダ全体の完全なHTTP／1バイト（例えば、「Content-Encoding: gzip」）、値がセンシティブであり、ログに記録されるべきではなく、決してインデックス付けされるべきではないか否か（HTTP／2）、および値が変化しており、キャッシュもインデックス付けもされるべきではないか否か（HTTP／2）のすべてをキャッシュすることができる。

【0071】

一実施形態によれば、要求からパースされたヘッダに対して、システムは、サーバまたは他のプロセスが実際にヘッダ値を要求するまで、バイトを格納することを含む遅延評価を使用することができ、通常、決して要求されることのないヘッダがあるため、これにより、各要求を処理するために使用されるプロセッサ（CPU）リソースが低減する。

【0072】

一実施形態によれば、既知のヘッダ名（すなわち、一般的に使用されるヘッダ名）は、列挙（HeaderEnum）で実装される。列挙は、HTTP／2によって、文字列の頻繁な小文字化を防ぐために使用される、小文字化された名前（String）、および、HTTP／1によって、予期されるヘッダ名を有するために使用される、「デフォルトケース」の名前（String）、ヘッダが存在するか否かを迅速に識別するため、および配列へのインデックス付けされたアクセスを通してそのようなヘッダの検索を最適化するための、要求および応答において最適化されたストレージをサポートするためのヘッダのインデックスを維持する。これは著しい改善であり、その理由は、さまざまな実装では、本来はヘッダを格納するためにマップを使用する場合があり、マップの格納および検索の動作は、インデックス付き配列への直接アクセスよりも著しくコストがかかるためである。

【0073】

図10は、一実施形態によるHTTPヘッダ処理に対するサポートをさらに示す。

【0074】

図10に示すように、一実施形態によれば、名前API262（例えば、HeaderNameインターフェース）および値API264（例えば、HeaderValueインターフェー

10

20

30

40

50

ス)として提供することができる高レベルAPIを介して、ヘッダオブジェクト260にアクセスすることができる。

【0075】

一実施形態によれば、HeaderNameインターフェースは以下のようなメソッドを含むことができる。

【0076】

メソッドString lowerCase()—HTTP/2のヘッダの名前 (content-typeなど)

。

【0077】

メソッドString defaultCase()—HTTP/1のヘッダの名前 (Content-Typeなど)。

10

【0078】

メソッドint index()—「既知の」ヘッダ (0+) およびカスタムヘッダ (−1) を識別するため。

【0079】

メソッドboolean isPseudoHeader()—HTTP/2 疑似ヘッダ、例えば、HTTP/1の「Host」は、HTTP/2における「:authority」疑似ヘッダである。

【0080】

一実施形態によれば、HeaderValueインターフェースは以下のようなメソッドを含むことができる。

20

【0081】

メソッドHeaderName headerName()—ヘッダの名前。

【0082】

メソッドString Value()—ヘッダの最初の値、遅延評価によって裏付けられる可能性がある。

【0083】

多値ヘッダに対する他のメソッド。

【0084】

図11は、一実施形態によるHTTPヘッダ処理に対するサポートをさらに示す。

【0085】

図11に示すように、一実施形態によれば、HTTPヘッダは、オブジェクト型として使用される。既知のヘッダ282を有するかまたはそのヘッダ名284によって示されるような要求280が作成されると、システムは、値286について、値のデータバイト (単一値の場合)、ヘッダ全体の完全なHTTP/1バイト (例えば「Content-Encoding: gzip」)、値がセンシティブであり、ログに記録されるべきではなく、決してインデックス付けされるべきではないか否か (HTTP/2)、および値が変化しており、キャッシュもインデックス付けもされるべきではないか否か (HTTP/2) のすべてをキャッシュすることができる。

30

【0086】

図12は、一実施形態によるHTTPヘッダ処理に対するサポートをさらに示す。

40

【0087】

図12に示すように、一実施形態によれば、要求からパースされたヘッダに対して、システムは、サーバまたは他のプロセスが実際にヘッダ値を要求するまで、遅延評価を使用し、バイトをキャッシュされたヘッダ値288として格納することができる。

【0088】

一実施形態によれば、システムは、例えば以下のプログラムによるなど、さまざまなインターフェース、クラス、またはメソッドのサポートを提供することができ、このプログラムは、例えば以下のように、必要な場合にのみバイトから作成される文字列の実装 (およびキャッシュ値) を提供する：

```
inner class Header
```

50

```

constants on class
methods
    create(String)
    create(String, String)
    createFromLowercase(String)
inner class HeaderName
inner class HeaderValue
    methods
        createCached(HeaderName, String)
        create(HeaderName, LazyString)
        create(HeaderName, String...)
        createCached(HeaderName, boolean, boolean, String)
        create(HeaderName, boolean, boolean, String...)
inner class HeaderValues
    constants on class

```

10

20

30

40

50

【0089】

上記の例は、例示の目的で提供するものであり、さまざまな実施形態によれば、インターフェース、クラス、またはメソッドの他の例を提供することができる。

【0090】

図13は、一実施形態による、HTTPヘッダを処理するための、マイクロサービス環境で使用される方法を示す。

【0091】

図13に示すように、一実施形態によれば、ステップ290において、1つまたは複数のプロセッサおよびメモリを含むコンピュータにおいて、クライアントアプリケーションおよびサーバアプリケーションがマイクロサービス環境内で通信することを可能にする、（例えば、マイクロサービスライブラリによって提供されるような）マイクロサービス環境が提供される。

【0092】

ステップ292において、システムは、HTTPヘッダ処理コンポーネントまたは機能性に関連して、ヘッダ値を格納するキャッシュと、複数のヘッダ型の各々について、そのヘッダ型のHTTP要求および応答を処理する際に使用される、1つまたは複数のヘッダ型名および関連付けられたインデックスを格納するヘッダ列挙とを提供する。

【0093】

ステップ294において、HTTP要求に関連付けられたヘッダ名および／またはヘッダ値を設定または判定する際に使用される、名前アプリケーションプログラムインターフェース（API）および値APIとともに、ヘッダオブジェクトが提供される。

【0094】

ステップ296において、既知のヘッダ型を有する要求を受け取ると、システムは、サーバプロセスによって要求された場合の後の遅延検索および評価のために、ヘッダ値をキャッシュに格納する。

【0095】

インターフェース例

一実施形態によれば、HTTPヘッダを提供するインターフェース例を以下に提供する。他の実施形態によれば、他のタイプのインターフェースを提供することができる。

【0096】

インターフェースHeaderName

一実施形態によれば、HTTPヘッダ名を提供するためにHttp.HeaderNameインターフェースを使用することができ、それは、例えば以下のようなさまざまなメソッドを含むことができる。

【0097】

メソッドString lowerCase(): HTTP / 2で使われるこのヘッダの小文字値は、HTTP / 1によるルックアップのために使われる場合がある。この値のバリデーションはないため、これが大文字を含む場合、挙動は未定義である。一実施形態では、このメソッドは、小文字のヘッダ名を返す。

【0098】

メソッドString defaultCase(): HTTP / 1で使われるようなヘッダ名、またはこのヘッダの「人間可読」値。一実施形態では、このメソッドは、ヘッダの名前を返し、大文字および/または小文字を使用することができる。

【0099】

メソッドint index(): このヘッダのインデックス（既知のインデックス付きヘッダのうちの1つである場合）、またはこれがカスタムヘッダ名である場合は-1。一実施形態では、このメソッドは、このヘッダのインデックスを返す。

10

【0100】

メソッドboolean isPseudoHeader(): HTTP / 2は、疑似ヘッダを「:」文字で始まるヘッダとして定義する。これらは、HTTP / 1の（パス、権限などを定義するための）プロローグ行の代わりに、および応答のステータス行の代わりに使われる。一実施形態では、このメソッドは、このヘッダが疑似ヘッダであるか否かを返す。

【0101】

インターフェースHeaderValue

一実施形態によれば、HTTPヘッダ値を提供するためにHttp.HeaderValueインターフェースを使用することができ、それは、例えば以下のようなさまざまなメソッドを含むことができる。

20

【0102】

メソッドstatic Http.HeaderValue createCached(Http.HeaderName name, String value)。バイト値を作成してキャッシュする。このメソッドは、ヘッダ値が定数に格納されているか、または繰り返し使われる場合に使われる。提供することができるパラメータとしては、例えば、name-ヘッダの名前、value-ヘッダの値が挙げられる。一実施形態では、このメソッドは、新たなヘッダを返す。

【0103】

メソッドstatic Http.HeaderValue create(Http.HeaderName name, LazyString value)。単一の値を有する新たなヘッダを作成する。このヘッダは、不変でありセンシティブではないとみなされる。提供することができるパラメータとしては、例えば、name-ヘッダの名前、value-値を有する遅延文字列が挙げられる。一実施形態では、このメソッドは、新たなヘッダを返す。

30

【0104】

メソッドstatic Http.HeaderValue create(Http.HeaderName name, String value)。単一の値を有する新たなヘッダを作成する。このヘッダは、不変でありセンシティブではないとみなされる。提供することができるパラメータとしては、例えば、name-ヘッダの名前、value-ヘッダの値が挙げられる。一実施形態では、メソッドは、新たなヘッダを返す。

40

【0105】

メソッドstatic Http.HeaderValue create(Http.HeaderName name, String... values)。新たなヘッダを作成する。このヘッダは、不変でありセンシティブではないとみなされる。提供することができるパラメータとしては、例えば、name-ヘッダの名前、values-ヘッダの値が挙げられる。一実施形態では、このメソッドは、新たなヘッダを返す。

【0106】

メソッドstatic Http.HeaderValue create(Http.HeaderName name, List<String> values)。新たなヘッダを作成する。このヘッダは、不変でありセンシティブではないとみなされる。提供することができるパラメータには、例えば、name-ヘッダ

50

の名前、`values`-ヘッダの値がある。一実施形態では、このメソッドは、新たなヘッダを返す。

【0107】

メソッド `static Http.HeaderValue createCached(Http.HeaderName name, boolean changing, boolean sensitive, String value)`。バイト値を作成し、キャッシュする。このメソッドは、ヘッダ値が定数に格納されるか、または繰り返し使用される場合に使用される。提供することができるパラメータには、例えば、`name`-ヘッダ名、`changing`- (`HTTP/2` に対するキャッシュを無効にするように) 値が頻繁に変化しているか否か、`sensitive`- (`HTTP/2` に対するキャッシュを無効にするように) 値がセンシティブであるか否か、`value`-ヘッダの値がある。一実施形態では、このメソッドは、新たなヘッダを返す。

10

【0108】

メソッド `static Http.HeaderValue create(Http.HeaderName name, boolean changing, boolean sensitive, String... values)`。新たなヘッダを作成する。提供することができるパラメータには、例えば、`name`-ヘッダの名前、`changing`- (`HTTP/2` に対するキャッシュを無効にするように) 値が頻繁に変化しているか否か、`sensitive`- (`HTTP/2` に対するキャッシュを無効にするように) 値がセンシティブであるか否か、`values`-ヘッダの値がある。一実施形態では、このメソッドは、新たなヘッダを返す。

【0109】

メソッド `String name()`。ユーザによって構成されるような、または通信で受信されるようなヘッダの名前。一実施形態では、このメソッドは、`HTTP/2` ヘッダでは常に小文字のヘッダ名を返す。

20

【0110】

メソッド `Http.HeaderName headerName()`。ヘッダのヘッダ名。一実施形態では、このメソッドは、ヘッダ名を返す。

【0111】

メソッド `String value()`。このヘッダの最初の値。一実施形態では、このメソッドは、最初の値を返す。

【0112】

メソッド `List<String> allValues()`。このヘッダのすべての値。一実施形態では、このメソッドは、構成されたすべての値を返す。

30

【0113】

メソッドの `default List<String> allValues(boolean split)`。このヘッダのすべての値。このヘッダがカンマ区切りの値を有する単一ヘッダとして定義されている場合、`split` を `true` に設定する。提供することができるパラメータには、例えば、`split`-単一の値をカンマで区切るべきか否か、値がすでにリストにある場合は何もしない、がある。一実施形態では、このメソッドは、値のリストを返す。

【0114】

メソッド `boolean sensitive()`。センシティブヘッダはログに記録されるべきでも、インデックス付けされるべきでもない (`HTTP/2`)。一実施形態では、このメソッドは、このヘッダがセンシティブであるか否かを返す。

40

【0115】

メソッド `boolean changing()`。変化するヘッダはキャッシュされるべきではなく、それらヘッダの値はインデックス付けされるべきではない (`HTTP/2`)。一実施形態では、このメソッドは、このヘッダの値が頻繁に変化しているか否かを返す。

【0116】

メソッド `default byte[] valueBytes()`。単一値のヘッダの値のキャッシュされたバイト。一実施形態では、このメソッドは、値のバイトを返す。

【0117】

50

メソッド `default void writeHttp1Header(BufferData buffer)`。現ヘッダを HTTPヘッダとして提供されたバッファに書き込む。提供することができるパラメータには、例えば、`buffer`（拡張するべき）書き込むためのバッファ。

【0118】

ヘッダの実装

一実施形態によれば、要求からパースされたか、または後の検索のためにユーザによって(処理するためにユーザによって、またはネットワークを介して書き出すためにサーバによって)構成された、ヘッダを格納するために、ヘッダオブジェクトを使用することができる。以下の例は、ヘッダを取得および設定するときに直接アクセスを利用するために必要な部分を示す。

```
class Headers
```

```
//インデックスを介してアクセスされる、既知のヘッダの配列
```

```
HeaderValue[] knownHeaders = new HeaderValue[KNOWN_HEADER_SIZE]
```

```
Map<HeaderName, HeaderValue> customHeaders = new Map()
```

```
HeaderValue find(HeaderName name)
```

```
    var index = name.index()
```

```
    return index == -1 ? customHeaders.get(name) : knownHeaders[index]
```

```
void set(HeaderValue value)
```

```
    var name = header.headerName()
```

```
    var index = name.index()
```

```
    if (index == -1) {
```

```
        customHeaders.put(name, value)
```

```
    } else {
```

```
        knownHeaders[index] = value
```

```
    }
```

【0119】

B. クライアントおよびサーバ接続の抽象化

一実施形態によれば、`loom`などの要求処理コンポーネントは、仮想スレッドを使用してクライアント-サーバ通信の並行性をサポートする。このような仮想スレッドによって、クライアントHTTPアプリケーションは、通常のブロッキング方式で動作する、例えばサーバプロセスへの接続を要求することができ、一方で、基礎となる要求処理（例えば、`loom`ベースの）環境は、要求処理の並行性を処理する。

【0120】

このような要求処理コンポーネントを用いてクライアントアプリケーションを開発する能力は、より単純なコード開発プロセスを提供する。例えば、アプリケーション開発者は、自身のアプリケーション自体にリアクティブプログラミングを組み込む必要がない。

【0121】

一実施形態によれば、ソフトウェアアプリケーションまたはサービスのテストをサポートするために、クライアント接続は、テストの目的で、例えばHTTPを呼び出さないが、代わりにサーバ接続に逆関数を公開するインメモリ実装に置き換えることができ、例えば、クライアント接続のデータライタに書き込まれたデータバイトは、サーバ接続においてデータリーダを通して利用可能であり、その逆も同様である。

【0122】

テスト中、インメモリ実装は、メモリ内のブロッキングキューを使用し、HTTPプロトコルの使用をエミュレートするが実際のソケットを使用しない。このようなテスト（例えば、単体テスト）が完了すると、クライアントおよび／またはサーバコードを、実際のネットワークソケットに対してテストすることができる（例えば、統合テスト）。

【0123】

例えば、アプリケーション開発者が単体テストを実行するか統合テストを実行するかに応じて、同じテストを書き直す必要なしに、容易にテストすることができるサービスを作

10

20

30

40

50

成することが望まれる場合がある。通常、例えばイベントループ機構、リアクティブコードなどを使用して、パフォーマンスを最適化するため、抽象化は極めて複雑になる。

【0124】

説明したアプローチは、テストまたはデバッグの改善を可能にし、例えば、テストまたはデバッグ中、スレッドダンプはトラブルシューティングの目的でより有用な情報を提供する。

【0125】

図14は、一実施形態による、クライアントおよびサーバの抽象化に対するサポートを示す。

【0126】

図14に示すように、一実施形態によれば、ソフトウェアアプリケーションまたはサービスのテストをサポートするために、クライアント接続は、テストの目的で、例えばHTTPを呼び出さないが、代わりにサーバ接続に逆関数を公開するインメモリ実装300に置き換えることができ、例えば、クライアント接続のデータライタ304に書き込まれたデータバイトは、サーバ接続においてデータリーダ306を通して利用可能であり、その逆も同様である。

【0127】

図15は、一実施形態による、クライアントおよびサーバの抽象化に対するサポートをさらに示す。

【0128】

図15に示すように、一実施形態によれば、テスト中（例えば、サーバコード320に向けられた要求310のプロセスをテストする際）、インメモリ実装は、メモリ内のブロッキングキューを使用し、HTTPプロトコルの使用をエミュレートするが実際のソケットを使用しない。

【0129】

図16は、一実施形態による、クライアントおよびサーバの抽象化に対するサポートをさらに示す。

【0130】

図16に示すように、一実施形態によれば、インメモリ実装を使用してテスト（例えば、単体テスト、または統合テスト）が完了すると、要求312の処理に使用するために、クライアント330および／またはサーバコード322を実際のソケットに対してテストすることができる。

【0131】

図17は、一実施形態による、クライアントおよびサーバの抽象化に対するサポートをさらに示す。

【0132】

図17に示すように、一実施形態によれば、例えばサーバコード324を更新し、メモリ内のブロッキングキューを使用するインメモリ実装に対してテストを実行し、要求314の処理に使用するために、実際のソケットに対してクライアント332を実装するために、プロセスを繰り返すことができる。

【0133】

一実施形態によれば、システムは、接続を2つの基本動作に抽象化し、すなわち、（1）データライタ（DataWriter）を使用して、接続を書き込むことができ、（2）データリーダ（DataReader）を使用して、接続から読み出すことができる。

【0134】

一実施形態によれば、システムは、アプリケーション開発者が、クライアント接続を、ネットワークソケット通信を呼び出さないが、サーバ接続に逆関数を公開するインメモリ実装に置き換えることを可能にする（クライアント接続のデータライタに書き込まれたバイトは、サーバ接続においてデータリーダを介して利用可能であり、逆も同様である）。

【0135】

10

20

30

40

50

一実施形態によれば、これにより、アプリケーション開発者は、サーバコードをテストするための単一のアプリケーションコードを書くことができ、実際のまたは現実のHTTPクライアントとインメモリHTTPクライアントとを単に切り替えることによって、アプリケーション開発者は、単体テストにおけるルーティングコードの妥当性と、統合テストを使用する（例えば、ポートを開いてHTTPプロトコルでそれを呼び出す）サーバコード全体の妥当性との両方をテストすることができる。

【0136】

一実施形態によれば、これを使用するテストの例としては、インメモリ接続の抽象化を使用するHTTP1Clientを実装するDirectClientのインスタンスを作成する単体テスト、ネットワークを介して接続するHTTP1Clientのインスタンスを作成する統合テスト、またはHTTP1Clientのインスタンスを取得し、例えば以下のような実際のテストを定義する抽象クラスが挙げられる。

```
@Test
void testHealthObserver() {
    try (Http1ClientResponse response
        = client.get("/observe/health").request()) {
        assertThat(response.status(), is(HttpStatus.NO_CONTENT_204));
    }
}
```

【0137】

一実施形態によれば、システムは、以下に例示するように、さまざまなインターフェース、クラス、またはメソッドのサポートを提供することができる。

【0138】

一実施形態によれば、バイトを提供することができる抽象化（DataReader）は、例えば以下のように、追加データを引き出すことができるデータリーダーとして提供することができる。

```
DataReader {
    /**
     * バイトのサプライヤからのデータリーダー。
     * @param bytesSupplier そのソース（ネットワーク、またはメモリ）に関わらず、
     * より多くのデータに対して引き出すことができるサプライヤ
     */
    DataReader(Supplier <byte[]> bytesSupplier)
    /**
     * 現在引き出されているデータで利用可能なバイトの数。
     * @return 利用可能なバイトの数
     */
    int available()
    /**
     * 次のデータを引き出す。
     */
    void pullData
    /**
     * nバイトスキップする。
     * @param lenToSkip（現在の容量以下でなければならない）スキップすべき
     * バイトの数
     */
    void skip(int lenToSkip)
    /**
```

```

    * 少なくとも1バイトが利用可能であることを確実にする。
    */
void ensureAvailable()
/**
    * 1バイトを読み出す
    * @return 次のバイト
    */
byte read()
/**
    * 次のバッファを読み出す。
    * {#available()}数のバイトをバッファに読み込み、ポジションを移動する。
    * @return 利用可能なバイトをラップしたバッファデータ
    */
BufferData readBuffer()
/**
    * 定義されたサイズの次のバッファを読み込む。長さが利用可能でない場合、
    * 追加のデータを引き出す。
    * ポジションを移動する。
    * @param length 読み込むべきデータの長さ
    * @return 要求された長さを有するバッファデータ
    */
BufferData readBuffer(int length)

```

10

20

30

40

50

【0139】

一実施形態によれば、バイトを消費することができる抽象化（DataWriter）は、例えば以下のように、基礎となるトランスポート（例えば、ソケット）にデータを書き込むことができるデータライタとして提供することができる。

```

interface DataWriter {
    /**
    * Write buffersは、書き込みを送らせる場合があり、
    * 異なるスレッドに書きこむ場合がある。
    * このメソッドは、複数の呼び出しを結合して基礎となるトランスポートへの
    * 単一の書き込みにすることもできる。
    * @param buffers 書き込むべきバッファ
    */
    void write(BufferData... buffers)
}

```

【0140】

一実施形態によれば、テストのためのカスタム接続の使用を可能にするクライアントの接続の抽象化（ClientConnection）は、例えば以下のように提供することができる。

```

interface ClientConnection {
    /**
    * 応答バイトを提供するデータリーダー。
    * @return この接続から読み込むべきリーダー
    */
    DataReader reader();
    /**
    * クライアント要求が書き込むデータライタ。
    * @return この接続に書き込むべきライタ
    */
    DataWriter writer();
}

```

}

【0141】

一実施形態によれば、インメモリクライアント接続の実装（DirectClientConnection）を提供することができ、このような接続は、例えば以下のように、ネットワークプロトコルをバイパスして、サーバサイドルーティングに直接ルーティングする。

```

class DirectClientConnection implements ClientConnection {
    Router router;
    DataReader clientReader;
    DataWriter clientWriter;
    DataReader serverReader;
    DataWriter serverWriter;
    DirectClientConnection(Router router)
        this.router = router;
        var serverToClient = new ArrayBlockingQueue<>(1024);
        var clientToServer = new ArrayBlockingQueue<>(1024);
        this.clientReader = reader(serverToClient);
        this.clientWriter = writer(clientToServer);
        this.serverReader = reader(clientToServer);
        this.serverWriter = writer(serverToClient);
    }
    @Override
    public DataReader reader() {
        return clientReader;
    }
    @Override
    public DataWriter writer() {
        return clientWriter;
    }
    private DataWriter writer(ArrayBlockingQueue<byte[]> queue) {
        return new DataWriter() {
            @Override
            public void write(BufferData... buffers) {
                for (BufferData buffer : buffers) {
                    write(buffer);
                }
            }
            write(BufferData buffer) {
                byte[] bytes = new byte[buffer.available()];
                buffer.read(bytes);
                try {
                    queue.put(bytes);
                } catch (InterruptedException e) {
                    throw new IllegalStateException("スレッドが中断された",
e);
                }
            }
        };
    }
    private DataReader reader(ArrayBlockingQueue<byte[]> queue) {
        return new DataReader(() -> {

```

```

        byte[] data;
        try {
            data = queue.take();
        } catch (InterruptedException e) {
            throw new IllegalArgumentException("スレッドが中断された",
e);
        }
        if (data.length == 0) {
            return null;
        }
        return data;
    });
}
}

```

10

【0142】

一実施形態によれば、インメモリ配列の代わりにネットワークソケットを有するリーダーおよびライタを使用するHTTP／1.1のネットワーク経由接続の実装は、例えば以下のように提供することができる。

```

class Http1ClientConnection implements ClientConnection {
    private DataReader reader;
    private DataWriter writer;
    private String host;
    private int port;
    @Override
    public DataReader reader() {
        return reader;
    }
    @Override
    public DataWriter writer() {
        return writer;
    }
    Http1ClientConnection connect() {
        try {
            socket = new Socket();
            socket.connect(new InetSocketAddress(host, port),
                (int) options.connectTimeout().toMillis());
        } catch (IOException e) {
            throw new UncheckedIOException(+ host + ":" + port "に接続で
きなかった", e);
        }
        var helidoSocket = createSocket(socket);
        // to wrap input and output streams
        this.reader = new DataReader(helidoSocket);
        this.writer = new DataWriter() {
            @Override
            public void write(BufferData... buffers) {
                for (BufferData buffer : buffers) {
                    write (buffer);
                }
            }
        }
    }
}

```

20

30

40

50

```

        void write(BufferData buffer) {
            helidonSocket.write(buffer);
        }
    };
    return this;
}
}

```

【0143】

一実施形態によれば、クライアントとサーバとの間の通信は、クライアントのために、さまざまなコンポーネント、すなわち、（上述したような）ClientConnection、ならびに要求／応答に基づくプロトコルのためのClientRequest、およびClientResponseに抽象化することができる。

10

【0144】

一実施形態によれば、クライアントとサーバとの間の通信は、サーバのために、さまざまなコンポーネント、すなわち、ServerConnection、ならびに要求／応答に基づくプロトコルのためのServerRequest、およびServerResponseに抽象化することができる。

【0145】

クライアントを実装するとき、開発者は、新たなClientRequestで使用する接続を選択することができ、ネットワークベースの接続またはインメモリ接続のいずれかを提供する。

20

【0146】

要求／応答パラダイムに基づかないクライアント（リスナーの概念を使用するWeb Socketプロトコルなど）を実装する場合、クライアントは抽象化を直接処理し、インメモリ接続が使用される場合はメモリ内のサーバリスナーを呼び出すか、またはサーバ側のリスナーを処理するためにネットワーク経由でサーバを呼び出す。インメモリ通信を使用する直接のWeb Socketクライアントは、例えば以下のように提供することができる。

```
DirectWsClient implements WsClient {
```

```
    WsRouting routing; // WebSocket ルーティング、ルーティング抽象化の発明を参照
```

30

```
    // 提供されたクライアントリスナーを使用してインメモリで接続する
```

```
    connect(URI uri, WsListener listener) {
```

```
        HttpPrologue prologue = HttpPrologue.create("ws", "ws", ...)
```

```
        WsRoute route = routing.findRoute(prologue);
```

```
        // 使用すべきWebSocketルートを発見する
```

```
        // クライアントリスナーとサーバルートとの間のインメモリ接続を作成する
```

```
        DirectWsConnection conn = DirectWsConnection.create(prologue, listener, route);
```

```
        conn.start(); // start the communication
```

40

```
    }
```

```
}
```

【0147】

一実施形態によれば、ネットワーク経由のWeb Socketクライアント接続のための実装（WsClientImpl）を使用するメソッド接続は、例えば以下のように提供することができる。

```
DirectWsClient implements WsClient {
```

```
    connect(URI uri, WsListener listener) {
```

```
        Socket socket = new Socket();
```

```
        Socket.connect(uri.getHost(), uri.getPort());
```

50

```

var helidonSocket = helidonSocket(socket);
handleUpgradeToWs(helidonSocket);
ClientWsConnection session = new ClientWsConnection(listener,
    helidonSocket);
// VirtualThreadエグゼキュータなどのエグゼキュータで
// クライアント／サーバセッションを実行する
executor().submit(session);
}
}

```

10

【0148】

一実施形態によれば、サーバ接続で使用するべきデータライタおよびデータリーダーを使用するサーバ側の同等物（ConnectionContext）は、例えば以下のように提供することができる。

```

public interface ConnectionContext extends SocketContext {
    /**
     * 応答バイトを書きこむべきデータライタ。
     * @return データライタ
     */
    DataWriter dataWriter();
    /**
     * 要求バイトを読み込むべきデータリーダー。
     * @return データリーダー
     */
    DataReader dataReader();
    /**
     * 異なるタイプのルーティング（HTTP、WebSocket、grpc）を含む可能性のあるルータ。
     * @return ルータ
     */
    Router router();
}

```

20

30

【0149】

上記の例は、例示を目的として提供するものであり、さまざまな実施形態によれば、インターフェース、クラス、またはメソッドの他の例を提供することができる。

【0150】

図18は、一実施形態による、クライアントおよびサーバの抽象化を提供するための、マイクロサービス環境で使用される方法を示す。

【0151】

図18に示すように、一実施形態によれば、ステップ340において、1つまたは複数のプロセッサおよびメモリを含むコンピュータにおいて、クライアントアプリケーションおよびサーバアプリケーションがマイクロサービス環境内で通信することを可能にする、（例えば、マイクロサービスライブラリによって提供されるような）マイクロサービス環境が提供される。

40

【0152】

ステップ342において、システムは、クライアント接続の代わりに、インメモリ実装が逆関数を例えばHTTPのサーバ接続に公開し、クライアント接続のデータライタ（DataWriter）に書き込まれたデータバイトがサーバ接続においてデータリーダー（DataReader）を通して利用可能になる、クライアントサーバ接続抽象化コンポーネントまたは機能性を提供する。

【0153】

50

ステップ 3 4 4 において、ユーザは、例えば HTTP のクライアントで使用されるように、サーバコードをテストするためのルーティングコードを準備し、ルーティングコードの妥当性をテストするために、実際のまたは現実のクライアントとクライアントのインメモリ実装との間で切り替えることができる。

【 0 1 5 4 】

ステップ 3 4 6 において、ユーザは、ルーティングコードおよび／またはサーバコードを更新し、テストを続けるために実際のまたは現実のクライアントとインメモリ実装との間で切り替え続けることができる。

【 0 1 5 5 】

インターフェース例

一実施形態によれば、ClientConnection インターフェース（その例は上記で説明している）を使用して、テストのためのカスタム接続の使用を可能にするクライアント接続を提供することができる。

【 0 1 5 6 】

C. ルータの抽象化

異なる通信プロトコルを介してクライアントーサーバ通信をサポートするクラウド環境内では、各通信プロトコルが、メッセージヘッダを含む、参加者間で渡されるメッセージのフォーマット化に関する独自の要件を有する場合がある。

【 0 1 5 7 】

例えば、HTTP / 1 プロトコルと HTTP / 2 プロトコルとは、それらのプロトコルを介して要求または応答メッセージを送信するときに使用されるメッセージヘッダのフォーマットに関して、異なる要件を有する。

【 0 1 5 8 】

一実施形態によれば、システムは、例えば HTTP 1、gRPC over HTTP 2、および／または WebSocket などの複数のネットワークまたは通信プロトコルを介して要求に対応するように動作することができるサーバの構成を可能にする。

【 0 1 5 9 】

ルートの抽象化によって、各プロトコルがそれ自体のルーティング、例えば HTTP ルーティングを有することができる。下位抽象化が、例えば HTTP 1 ルートまたは HTTP 2 ルートなど、より具体的なルートを提供することができる。特定のプロトコルを介して受信された接続／要求に対して、システムは、その要求に適切なルートを選択し、それに応じて処理することができる。

【 0 1 6 0 】

例えば、各プロトコルの実装に依存することなく、サポートされているあらゆるプロトコルを処理することができるサーバビルダ上でルーティングを構成することが望ましい場合がある。このためには、アプリケーション開発者が新たなルーティングをサーバに追加することができるようにする、ルータおよびさまざまなルーティングの抽象化が必要である。例えば Java 環境では、クラスパスにライブラリを追加するだけで、このような抽象化を提供することができる。他の実装では、複数のプロトコルのバージョンで HTTP ルーティングをサポートする場合があるが、バージョン固有のルートを作成する機能がない可能性がある。さらに、実装によっては、ユーザが必要とするプロトコルバージョンに応じた特定の処理、またはこれらのプロトコルに対する明示的な依存を必要とするものもある。

【 0 1 6 1 】

説明したアプローチによって、複数のプロトコルをサポートするサーバの、定義が容易な構成が可能になる。

【 0 1 6 2 】

図 1 9 は、一実施形態による、ルータの抽象化に対するサポートを示す。

【 0 1 6 3 】

図 1 9 に示すように、一実施形態によれば、システムは、複数のネットワークまたは通

10

20

30

40

50

信プロトコルを介して要求に対応するように動作することができるサーバの構成 350 を可能にし、これらのプロトコルの各々は、サーバコード 360 およびビルダ 362 で使用するために、ルーティングタイプ 352、例えばメソッド 355 を有するルーティングタイプ A 354 に関連付けられる。

【0164】

ルートの抽象化により、各プロトコルが、独自のルーティング、例えば HTTP ルーティングを有することができる。下位抽象化が、例えば HTTP / 1 ルートまたは HTTP / 2 ルートなど、より具体的なルートを提供することができる。特定のプロトコルを介して受信された要求メッセージに対して、システムは、要求に適切なルートを選択し、それに従って要求を処理することができる。

10

【0165】

図 20 は、一実施形態による、ルータの抽象化に対するサポートをさらに示す。

【0166】

図 20 に示すように、一実施形態によれば、ビルダは、サーバコード内で第 1 のプロトコル A に対してプロトコルサポート 364 を提供することができる。システムは同様に、サーバコードとともに使用するために、メソッド 359 を有するルーティングタイプ B 358 をサポートするサーバの構成を可能にする。

【0167】

図 21 は、一実施形態による、ルータの抽象化に対するサポートをさらに示す。

【0168】

図 21 に示すように、一実施形態によれば、ビルダは、サーバコード内で第 2 のプロトコル B に対してプロトコルサポート 366 を提供することができる。

20

【0169】

図 22 は、一実施形態による、ルータの抽象化に対するサポートをさらに示す。

【0170】

図 22 に示すように、一実施形態によれば、構成されるようなサーバは、例えば、プロトコル A 372 を使用する要求 A 370、およびプロトコル B 382 を使用する要求 B 380 など、複数のネットワークまたは通信プロトコルを介して要求に対応するように動作することができる。

【0171】

例えば、一実施形態によれば、このルーティングの抽象化は、HTTP バージョン固有のルートを作成する能力により、任意の HTTP バージョンのためのルーティングをサポートする。HTTP のサブセット (gRPC など)、またはさらには HTTP と無関係である、例えば (WebSocket など) HTTP からアップグレードした、他のタイプのプロトコルのためのルートを追加することができる。

30

【0172】

一実施形態によれば、これは、複数のタイプのルーティングを含むことができるルータ (Router) を抽象化することによって達成される。サーバをセットアップするとき、アプリケーション開発者は、そのビルダにルーティングを登録することができる。そのため、ルーティングは、そのプロトコルの構成を可能にするプロトコル固有のメソッドを含む。これらを組み合わせることで、例えば以下のように、単一のルータ構成を使用して、サポートされているすべてのプロトコルへの要求に対応することができる単一のサーバが作成される。

40

```
.addRouting(HttpRouting.builder()
.get("/", (req, res) -> res.send("Hello World"))
.route(Http1Route.route(GET, "/http1", (req, res)
-> res.send("Hello World 1")))
.route(Http2Route.route(GET, "/http2", (req, res)
-> res.send("Hello World 2"))))
.addRouting(GrpcRouting.builder())
```

50

```
.unary(Strings.getDescriptor(),
"StringService",
"Upper",
ProtocolsMain::grpcUpper))
.addRouting(WebsocketRouting.builder()
.endpoint("/websocket/echo", ProtocolsMain::wsEcho))
```

【0173】

一実施形態によれば、説明したアプローチのさまざまな利点としては、例えば、同じサーバ内でプロトコルを組み合わせる能力を挙げることができる。プロトコルは別個のモジュールであり、ルーティングAPIから各プロトコル実装への依存性はない（例えば、サーバに変更を加えることなく、追加のプロトコルおよびそれらのルートを追加する）。任意のHTTPバージョンによって対応するHTTPルート（上記に示す例では「/」ルート）、またはHTTPバージョン固有のルート（上記に示す例では「/http1」および「/http2」ルート）を有するHTTPルートを作成することができる。このアプローチは、ユーザからHTTPバージョンの完全な抽象化を提供し、すなわち、コードは、完全にバージョンに依存しないものとすることができる。このアプローチはまた、拡張可能なメカニズムも提供し、そのため、例えばHTTP/3をサポートするために、ルーティングコードを変更する必要はないはずである。

【0174】

一実施形態によれば、システムは、以下に例示するように、さまざまなインターフェース、クラス、またはメソッドのサポートを提供することができる。

【0175】

一実施形態によれば、ルータインタフェース（Router）は、ルーティングをセットアップするために使用することができ、2つ以上のプロトコルルーティングを含むことができる。メソッドは、例えば、特定のタイプのルーティングを得るためのrouting(Class <T>, T)を含むことができ、サーバ用のルータは、さまざまなタイプのルーティングを含み、そのため、サーバは、各プロトコルを可能な限り高速に処理することができる。例えば、以下の通りである。

```
public interface Router {
    /**
     * 特定のタイプのルーティングを取得する。
     * @param routingType ルーティングのタイプ
     * @param defaultValue ルーティングがこのルータにおいて定義されていない
     * 場合に
     * 使用すべきデフォルト値
     * @param <T> ルーティングのタイプ
     * @return 定義されたルーティング、または見つからない場合はデフォルト値
     */
    <T extends Routing> T routing(Class <T> routingType, T defaultValue);
    /**
     * スタンドアローンルータのためのビルダ。
     */
    interface Builder {
        /**
         * このルータに新たなルーティングを追加する
         * @param routing {@code webserver.http.HttpRouting}など、
         * 追加すべきルーティング
         * @return 更新されたビルダ
         */
```

10

20

30

40

50

```

        B addRouting(Routing routing);
    }
}

```

【0176】

一実施形態によれば、ルーティングマーカークインターフェース（Routing）は、例えば、以下のように提供することができる。

```

/**
 * ルーティングの抽象化。
 */
public interface Routing {
}

```

10

【0177】

一実施形態によれば、ルーティングが任意のHTTPバージョンを処理することができる、HTTPルーティング実装（HttpRouting）は、例えば以下のように提供することができる。

```

public final class HttpRouting implements Routing {
    private final ServiceRoute rootRoute;
    private HttpRouting(Builder builder) {
        this.rootRoute = builder.rootRules.build();
    }
}

```

20

```

/**
 * {io.webserver.http.HttpRouting.Builder router builder}の新たなインスタンスを作成する。

```

```

 * @return 新たなインスタンス
 */

```

```

public static Builder builder() {
    return new Builder();
}

```

```

/**
 * 要求をルーティングする。
 * ハンドラHTTPは、フィルタリングし、ルートを見つける。
 * @param ctx コンテキスト
 * @param request ルーティング要求
 * @param response ルーティング応答
 */

```

30

```

public void route(ConnectionContext ctx,
    RoutingRequest request,
    RoutingResponse response) {
    HttpPrologue prologue = request.prologue();
    RoutingResult result = RoutingResult.ROUTE;
    int counter = 0;
    while (result == RoutingResult.ROUTE) {

```

40

```
        counter++;

```

```
        if (counter == 10) {

```

```
            LOGGER.log(System.Logger.Level.ERROR,

```

```
            "10回超えて再ルーティングされた。これ以上ルーティングは試みな

```

```
            い");

```

```
            throw HttpException.builder()

```

```
                .request(HttpSimpleRequest.create(prologue,
                    request.headers()))

```

50

```

        .type(SimpleHandler.EventType.INTERNAL_ERROR)
        .build();
    }
    result = doRoute(ctx, request, response);
}
// 終了および完了
if (result == RoutingResult.FINISH) {
    return;
}
throw HttpException.builder()
    .request(request)
    .response(response)
    .type(SimpleHandler.EventType.NOT_FOUND)
    .message("エンドポイントが見つからない")
    .build();
}
private RoutingResult doRoute(ConnectionContext ctx,
    RoutingRequest request,
    RoutingResponse response) {
    HttpPrologue prologue = request.prologue();
    RouteCrawler crawler = rootRoute.crawler(ctx, request);
    while (crawler.hasNext()) {
        response.resetRouting();
        RouteCrawler.CrawlerItem next = crawler.next();
        request.path(next.path());
        try {
            next.handler().handle(request, response);
            if (response.shouldReroute()) {
                if (response.isSent()) {
                    return RoutingResult.FINISH;
                }
                HttpPrologue newPrologue =
                    response.reroutePrologue(prologue);
                request.prologue(newPrologue);
                response.resetRouting();
                return RoutingResult.ROUTE;
            }
            if (response.isNexted()) {
                if (response.isSent()) {
                    return RoutingResult.FINISH;
                }
                continue;
            }
            if (!response.hasEntity()) {
                // 次の要素を返さず、再ルーティングせず、単にそれを送信する！
                response.send();
            }
            return RoutingResult.FINISH;
        } catch (CloseConnectionException | HttpException e) {
            throw e;
        }
    }
}

```

```

    } catch (Exception thrown) {
        if (thrown.getCause() instanceof SocketException se) {
            throw new UncheckedIOException(se);
        }
        if (!response.isSent()) {
            boolean keepAlive = true;
            try {
                request.content().consume();
            } catch (Exception e) {
                keepAlive = request.content().consumed();
            }
            // 要求を消費することができなかったため、
            // 接続をクローズしなければならない
            throw HttpException.builder()
                .type(SimpleHandler.EventType.INTERNAL_ERR
OR)
                .cause(thrown)
                .request(request)
                .response(response)
                .message(thrown.getMessage())
                .setKeepAlive(keepAlive)
                .build();
        }
    }
    return RoutingResult.NONE;
}
private enum RoutingResult {
    ROUTE,
    FINISH,
    NONE
}
/**
 * {io.webserver.http.HttpRouting}に対するFluent APIビルダ
 */
public static class Builder implements HttpRules,
    common.Builder<Builder, HttpRouting> {
    private final ServiceRules rootRules = new ServiceRules();
    private Builder() {
    }
    @Override
    public HttpRouting build() {
        return new HttpRouting(this);
    }
    @Override
    public Builder route(HttpRoute route) {
        rootRules.route(route);
        return this;
    }
    @Override

```

```

public Builder route(Http.Method method, String pathPattern,
    Handler handler) {
    return route(HttpRoute.builder()
        .methods(method)
        .path(pathPattern)
        .handler(handler));
}

```

```

/**

```

```

 * get routeを追加する。
 * @param pathPattern パスパターン
 * @param handler 使用すべきハンドラ
 * @return 更新されたビルダ
 */

```

```

public Builder get(String pathPattern, Handler handler) {
    return route(HttpRoute.builder()
        .methods(Http.Method.GET)
        .path(pathPattern)
        .handler(handler));
}

```

```

}

```

【0178】

一実施形態によれば、HTTPサービスの抽象化（HttpService）は、例えば以下の
ように、単一ユニットを形成するルートの論理的なグループ化を提供する。

```

/**

```

```

 * {HttpRouting routing}ルールおよび関連するロジックのセットをカプセル化する。
 * <p>
 * {HttpRouting.Builder#register(java.util.function.Supplier[])}メソッドを
 * 使用して
 * {HttpRouting routing}にインスタンスを割り当てることができる。
 */

```

```

@FunctionalInterface

```

```

public interface HttpService {

```

```

    /**

```

```

 * このサービスのハンドラを追加するためにルーティングを更新する。
 * @param 更新すべきルール
 */

```

```

    void routing(HttpRules rules);
}

```

【0179】

一実施形態によれば、HTTPルートの抽象化（HttpRoute）は、例えば以下のよう
に、パス表現と、HTTPメソッド述語と、（任意のHTTPプロトコルバージョンによ
って使用可能であるべきである）基本的なHTTPルートを含む、一致する要求を処理す
るハンドラとの組み合わせを提供する。

```

public interface HttpRoute extends Route {

```

```

    /**

```

```

 * 新たなHTTPルートをビルドすべきビルダ。
 * @return ビルダ
 */

```

```

    static HttpRouteImpl.Builder builder() {
        return new Builder();
    }
}

```

10

20

30

40

50

```

}
/**
 * このルートが、提供された要求を受け入れるか否か。
 * @param prologue 要求のプロローグ
 * @return バリデーションの結果
 * @see PathMatchers.MatchResult#notAccepted()
 */
PathMatchers.MatchResult accepts(HttpPrologue prologue);
/**
 * このルートのハンドラ。
 * @return ハンドラ
 */
Handler handler();
/**
 * {io.webserver.http.HttpRoute}のためのFluent APIビルダ。
 */
class Builder implements common.Builder<Builder, HttpRoute> {
    private Predicate<Http.Method> methodPredicate =
        Http.Method.predicate();
    private PathMatcher pathMatcher = PathMatchers.any();
    private Handler handler;
    private Builder() {
    }
    @Override
    public HttpRoute build() {
        Objects.requireNonNull(handler, "ハンドラを提供しなければならな
い");
        return new HttpRouteImpl(this);
    }
    /**
     * このルートが処理すべきHTTPメソッド。
     * @param methods 処理すべきメソッド
     * @return 更新されたビルダ
     */
    public Builder methods(Http.Method... methods) {
        return methods(Http.Method.predicate(methods));
    }
    /**
     * 使用すべきメソッド述語。
     * @param methodPredicate メソッド述語
     * @return 更新されたビルダ
     */
    public Builder methods(Predicate<Http.Method> methodPredica
te) {
        this.methodPredicate = methodPredicate;
        return this;
    }
    /**
     * 処理すべきパスパターン。
     * @param pathPattern パスパターン

```

```

    * @return 更新されたビルダ
    */
    public Builder path(String pathPattern) {
        return this.path(PathMatchers.create(pathPattern));
    }
    /**
    * パス进行处理すべきパスマッチャー。
    * @param pathMatcher パスマッチャー
    * @return 更新されたビルダ
    */
    public Builder path(PathMatcher pathMatcher) {
        this.pathMatcher = pathMatcher;
        return this;
    }
    /**
    * 使用すべきハンドラ。
    * @param handler ハンドラ
    * @return 更新されたビルダ
    */
    public Builder handler(Handler handler) {
        this.handler = handler;
        return this;
    }
    Predicate<Http.Method> methodPredicate() {
        return methodPredicate;
    }
    PathMatcher pathPredicate() {
        return pathMatcher;
    }
    Handler handler() {
        return handler;
    }
}
}

```

10

20

30

40

50

【0180】

一実施形態によれば、システムは、以下のように、複数の実装、例えばHTTP／1固有のルート実装(Http1Route)をサポートすることができる。

```

/**
 * HTTP／1固有のルートを作成する。
 * @param method 受け入れられたメソッド
 * @param path パスパターン
 * @param handler ハンドラ
 * @return 新たなHTTP／1．1固有のルート
 */
public static Http1Route route(Http.Method method, String path,
    Handler handler) {
    return new Http1Route(Http.Method.predicate(method),
        PathMatchers.create(path), handler);
}
@Override

```

```

public PathMatchers.MatchResult accepts(HttpPrologue prologue) {
    if (!prologue.protocolVersion().equals("1.1")) {
        return PathMatchers.MatchResult.notAccepted();
    }
    if (!methodPredicate.test(prologue.method())) {
        return PathMatchers.MatchResult.notAccepted();
    }
    return pathMatcher.match(prologue.uriPath());
}
@Override
public Handler handler() {
    return handler;
}
@Override
public void beforeStart() {
    handler.beforeStart();
}
@Override
public void afterStop() {
    handler.afterStop();
}
}

```

10

20

【0181】

同様に、一実施形態によれば、システムは、例えば以下のように、HTTP / 2 固有のルート実装 (Http2Route) をサポートすることができる。

```

/**
 * 新たなHTTP / 2 のみのルートを作成する。
 * @param method 処理すべきメソッド
 * @param path パスパターン
 * @param handler ハンドラ
 * @return 新たなルート
 */
public static Http2Route route(Http.Method method, String path,
    Handler handler) {
    return new Http2Route(Http.Method.predicate(method),
        PathMatchers.create(path), handler);
}
@Override
public PathMatchers.MatchResult accepts(HttpPrologue prologue) {
    if (!prologue.protocolVersion().equals("2.0")) {
        return PathMatchers.MatchResult.notAccepted();
    }
    if (!methodPredicate.test(prologue.method())) {
        return PathMatchers.MatchResult.notAccepted();
    }
    return pathMatcher.match(prologue.uriPath());
}
@Override
public Handler handler() {
    return handler;
}

```

30

40

50

```

    }
    @Override
    public void beforeStart() {
        handler.beforeStart();
    }
    @Override
    public void afterStop() {
        handler.afterStop();
    }
}

```

10

【0182】

一実施形態によれば、WebSocket固有のルーティングを提供するWebSocketルーティング実装（WebSocketRouting）は、例えば以下のように提供することができる。

```

public class WebSocketRouting implements Routing {
    private final List<WebSocket> routes;
    private WebSocketRouting(Builder builder) {
        this.routes = new ArrayList<> (builder.routes);
    }
    /**
     * WebSocketルーティングのためのビルダ。
     * @return 新たなビルダ
     */
    public static Builder builder() {
        return new Builder();
    }
    WebSocket findRoute(HttpPrologue prologue) {
        for (WebSocket route : routes) {
            PathMatchers.MatchResult accepts = route.accepts(prologue);
            if (accepts.accepted()) {
                return route;
            }
        }
        return null;
    }
    /**
     * {websocket.webserver.WebSocketRouting}のためのFluent APIビルダ。
     */
    public static class Builder implements common.Builder<Builder,
        WebSocketRouting> {
        private final List<WebSocket> routes = new ArrayList<> ();
        private Builder() {
        }
        @Override
        public WebSocketRouting build() {
            return new WebSocketRouting(this);
        }
        /**
         * エンドポイントを追加する。
         * @param path エンドポイントのパス

```

20

30

40

50

```

    * @param listener 使用すべきリスナー
    * @return 更新されたビルダ
    */
    public Builder endpoint(String path, WsListener listener) {
        return route(WebSocket.create(path, listener));
    }
    /**
    * エンドポイントを追加する
    * @param path エンドポイントのパス
    * @param listener リスナーサプライヤ
    * @return 更新されたビルダ
    */
    public Builder endpoint(String path, Supplier<WsListener> listener) {
        return route(WebSocket.create(path, listener));
    }
    private Builder route(WebSocket wsRoute) {
        routes.add(wsRoute);
        return this;
    }
}

```

10

20

【0183】

一実施形態によれば、上記は、例えば以下のように、WebSocketエンドポイントを構成するために、WsListener (WsListener) クラスを使用する。

```

public interface WsListener {
    /**
    * テキストフラグメントを受信する。
    * @param session WebSocketセッション
    * @param text 受信したテキスト
    * @param lastはこの最後のフラグメントである
    */
    void onMessage(WsSession session, String text, boolean last);
    /**
    * バイナリフラグメントを受信する。
    * @param session WebSocketセッション
    * @param buffer データを含むバッファ
    * @param lastはこの最後のフラグメントである
    */
    void onMessage(WsSession session, BufferData buffer, boolean last);
    /**
    * pingを受信した。
    * @param session WebSocketセッション
    * @param buffer データを含むバッファ
    */
    void onPing(WsSession session, BufferData buffer);
    /**
    * pongを受信した。
    * @param session WebSocketセッション
    * @param buffer データを含むバッファ
    */
}

```

30

40

50

```

    */
    void onPong(WsSession session, BufferData buffer);
    /**
     * closeを受信した。
     * @param session WebSocketセッション
     * @param status クローズステータス
     * @param reason クローズの理由
     */
    void onClose(WsSession session, int status, String reason);
    /**
     * エラーが発生した。
     * @param session WebSocketセッション
     * @param t キャッチされたthrowable
     */
    void onError(WsSession session, Throwable t);
    /**
     * セッションがオープンしている。
     * @param session WebSocketセッション
     */
    void onOpen(WsSession session);
}

```

【0184】

一実施形態によれば、gRPC固有のルーティングを提供するgRPCルーティング実装(GrpcRouting)は、例えば以下のように提供することができる。

```

public class GrpcRouting implements Routing {
    private final ArrayList<GrpcRoute> routes;
    private GrpcRouting(Builder builder) {
        this.routes = new ArrayList<>(builder.routes);
    }
    /**
     * 新たなルーティングビルダ。
     * @return 新たなビルダ
     */
    public static Builder builder() {
        return new Builder();
    }
    Grpc<?, ?> findRoute(HttpPrologue prologue) {
        for (GrpcRoute route : routes) {
            PathMatchers.MatchResult accepts = route.accepts(prologue);
            if (accepts.accepted()) {
                return route.toGrpc(prologue);
            }
        }
        return null;
    }
    /**
     * {GrpcRouting}のためのFluent APIビルダ。
     */
    public static class Builder implements common.Builder<Builder, GrpcRouting> {

```

```

private final List<GrpcRoute> routes = new LinkedList<>();
private Builder() {
}
@Override
public GrpcRouting build() {
    return new GrpcRouting(this);
}
/**
 * grpcサービスを構成する。
 * @param service 追加すべきサービス
 * @return 更新されたビルダ
 */
public Builder service(GrpcService service) {
    return route(GrpcServiceRoute.create(service));
}
/**
 * 単項ルート。
 * @param proto proto記述子
 * @param serviceName サービス名
 * @param methodName メソッド名
 * @param method このルート进行处理すべきメソッド
 * @param <ReqT> 要求タイプ
 * @param <ResT> 応答タイプ
 * @return 更新されたビルダ
 */
public <ReqT, ResT> Builder unary(Descriptors.FileDescriptor
    proto, String serviceName, String methodName,
    ServerCalls.UnaryMethod<ReqT, ResT> method) {
    return route(Grpc.unary(proto, serviceName, methodName,
        method));
}
/**
 * 双方向ルート。
 * @param proto proto記述子
 * @param serviceName サービス名
 * @param methodName メソッド名
 * @param method このルート进行处理すべきメソッド
 * @param <ReqT> 要求タイプ
 * @param <ResT> 応答タイプ
 * @return 更新されたビルダ
 */
public <ReqT, ResT> Builder bidi(Descriptors.FileDescriptor
    proto, String serviceName, String methodName,
    ServerCalls.BidiStreamingMethod<ReqT, ResT> method) {
    return route(Grpc.bidi(proto, serviceName, methodName, met
hod));
}
/**
 * サーバストリーミングルート。
 * @param proto proto記述子

```

```

    * @param serviceName サービス名
    * @param methodName メソッド名
    * @param method このルート进行处理すべきメソッド
    * @param <ReqT> 要求タイプ
    * @param <ResT> 応答タイプ
    * @return 更新されたビルダ
    */
    public <ReqT, ResT> Builder serverStream(Descriptors.FileDesc
ripor
        proto, String serviceName, String methodName,
        ServerCalls.ServerStreamingMethod <ReqT, ResT> method) {
        return route(Grpc.serverStream(proto, serviceName, methodN
ame,
            method));
    }
    /**
    * クライアントストリーミングルート。
    * @param proto proto記述子
    * @param serviceName サービス名
    * @param methodName メソッド名
    * @param method このルート进行处理すべきメソッド
    * @param <ReqT> 要求タイプ
    * @param <ResT> 応答タイプ
    * @return 更新されたビルダ
    */
    public <ReqT, ResT> Builder clientStream(Descriptors.FileDescr
iptor
        proto, String serviceName, String methodName,
        ServerCalls.ClientStreamingMethod <ReqT, ResT> method) {
        return route(Grpc.clientStream(proto, serviceName, methodN
ame,
            method));
    }
    private Builder route(GrpcRoute route) {
        routes.add(route);
        return this;
    }
}

```

10

20

30

40

50

【0185】

一実施形態によれば、gRPCサービス（GrpcService）は、例えば以下のように、単一のprotoファイルによってバックアップされる単一のgRPCサービスとして、提供することができる。

```

public interface GrpcService {
    /**
    * このサービスのproto記述子。
    * @return protoファイル記述子
    */
    Descriptors.FileDescriptor proto();
}

```

```

* サービス名、デフォルトはこのクラスの単純な名前である。
* @return サービス名
*/
default String serviceName() {
    return getClass().getSimpleName();
}
/**
* ルーティングを更新する。
* @param routing ルーティング
*/
void update(Routing routing);
/**
* サービス固有のルーティング（proto記述子は{GrpcService#proto()}によっ
て提供される）。
*/
interface Routing {
    /**
    * 単項ルート。
    * @param methodName メソッド名
    * @param method ルート进行处理すべきメソッド
    * @param <ReqT> 要求タイプ
    * @param <ResT> 応答タイプ
    * @return 更新されたルーティング
    */
    <ReqT, ResT> Routing unary(String methodName,
ServerCalls.UnaryMethod <ReqT, ResT> method);
    /**
    * 双方向ルート。
    * @param methodName メソッド名
    * @param method ルート进行处理すべきメソッド
    * @param <ReqT> 要求タイプ
    * @param <ResT> 応答タイプ
    * @return 更新されたルーティング
    */
    <ReqT, ResT> Routing bidi(String methodName,
ServerCalls.BidiStreamingMethod <ReqT, ResT> method);
    /**
    * サーバストリーミングルート。
    * @param methodName メソッド名
    * @param method ルート进行处理すべきメソッド
    * @param <ReqT> 要求タイプ
    * @param <ResT> 応答タイプ
    * @return 更新されたルーティング
    */
    <ReqT, ResT> Routing serverStream(String methodName,
ServerCalls.ServerStreamingMethod <ReqT, ResT> method);
    /**
    * クライアントストリーミングルート。
    * @param methodName メソッド名
    * @param method ルート进行处理すべきメソッド

```

```

    * @param <ReqT> 要求タイプ
    * @param <ResT> 応答タイプ
    * @return 更新されたルーティング
    */
    <ReqT, ResT> Routing clientStream(String methodName,
    ServerCalls.ClientStreamingMethod <ReqT, ResT> method);
}
}

```

【0186】

上記の例は、例示の目的で提供するものであり、さまざまな実施形態によれば、インターフェース、クラス、またはメソッドの他の例を提供することができる。 10

【0187】

図23は、一実施形態による、ルータの抽象化を提供するための、マイクロサービス環境で使用される方法を示す。

【0188】

図23に示すように、一実施形態によれば、ステップ390において、1つまたは複数のプロセッサおよびメモリを含むコンピュータにおいて、クライアントアプリケーションおよびサーバアプリケーションがマイクロサービス環境内で通信することを可能にする、（例えば、マイクロサービスライブラリによって提供されるような）マイクロサービス環境が提供される。 20

【0189】

ステップ392において、システムは、例えばHTTPの要求ルーティングで使用されるように、サーバに新たなルーティングを追加するためにクラスパス上にライブラリを追加することによってサーバコードの構成を可能にする、ルータおよびルーティングの抽象化を含む、ルータ抽象化コンポーネントまたは機能性を提供する。

【0190】

ステップ394において、サーバをセットアップするとき、サーバはそのビルダにルーティングを登録することができ、ルーティングはそのプロトコルの構成を可能にするプロトコル固有のメソッドを含み、ルータはルータに関連付けられた複数のプロトコルをサポートする。 30

【0191】

ステップ396において、構成されるようなサーバは、サーバによって構成されサポートされるようなプロトコルを使用して、クライアントからの要求を受信して処理するように動作する。

【0192】

インターフェース例

一実施形態によれば、説明の目的で、ルータの抽象化を提供するためのインターフェース例を以下に提供する。他の実施形態によれば、他のタイプのインターフェースを提供することができる。

【0193】

インターフェースルータ

一実施形態によれば、ルータ（Router）インターフェースは、サーバが各プロトコルを迅速に処理できるように、さまざまなタイプのルーティングを提供するために使用することができ、例えば以下のように、さまざまなメソッドを含むことができる。

【0194】

メソッドstatic Router.Builder builder()。ルータ用のビルダ。一実施形態では、このメソッドは新たなビルダを返す。

【0195】

メソッドstatic Router empty()。空のルータ。一実施形態では、このメソッドは新たな空のルータを返す。 50

【0196】

メソッド `routing` (`T extends Routing`) `T routing(Class <T> routingType, T defaultValue)`。特定のタイプのルーティングを取得する。提供することができる `Type` パラメータとしては、例えば、`T` ルーティングのタイプが挙げられる。提供することができるパラメータとしては、例えば、`routingType`—そのルーティングのタイプ、`defaultValue`—そのルーティングがこのルータで定義されていない場合に使用すべきデフォルト値が挙げられる。一実施形態では、このメソッドは、定義されたルーティング、または見つからない場合のデフォルト値を返す。

【0197】

D. 接続のプロトコルの識別

異なる通信プロトコルを介してクライアント—サーバ通信をサポートするクラウド環境内では、各通信プロトコルが、メッセージヘッダを含む、参加者間で渡されるメッセージのフォーマット化に関する独自の要件を有する場合がある。

【0198】

例えば、`HTTP / 1` プロトコルと `HTTP / 2` プロトコルとは、それらのプロトコルを介して要求メッセージまたは応答メッセージを送信するときに使用されるメッセージヘッダのフォーマットに関して、異なる要件を有する。

【0199】

一実施形態によれば、クライアント／サーバ接続の抽象化、ルータの抽象化、または単一ソケット上での異なるプロトコルの使用などの機能をサポートするために、拡張可能なサービスプロバイダインターフェース (SPI) は、要求メッセージおよび接続のプロトコル識別を可能にするソフトウェアメソッドを提供する。

【0200】

クライアントからサーバへの接続要求メッセージが受信されると、その要求を適切に処理することができるか否かに関して判定が行われる。プロトコル識別プロセスは、例えば、システムが要求プロトコルをサポートしていないこと、システムが、判断を行うのに十分なデータを有していないこと、またはそのような判定を行うために追加のデータが必要であることの標示を返すことができる。

【0201】

例えば、いくつかのシステムでは、サーバが接続要求を受信したときに、要求されている接続のタイプを知ることができない場合がある。本明細書で説明するように、このアプローチでは、異なるタイプの接続に対応する単一のサーバソケットを有することができる。通常、互換性のない接続タイプには異なるソケットを使用しなければならない。`HTTP / 2` では、サーバによっては、`HTTP / 1` からのアップグレードしかサポートしない場合があり、または、事前知識に対する要求を何らかの特別な `HTTP / 1` 要求にマッピングする場合がある。

【0202】

説明したアプローチでは、サポートされているいずれのプロトコルでもサーバを呼び出すことができ、サーバはそれを受け入れることができるか否かを自動的に判定する。

【0203】

図 24 は、一実施形態による、接続のプロトコルを識別するためのサポートを示す。

【0204】

図 24 に示すように、一実施形態によれば、クライアント／サーバ接続の抽象化、ルータの抽象化、または単一ソケット上での異なるプロトコルの使用などの機能をサポートするために、拡張可能なサービスプロバイダインターフェース (SPI) 402 は、要求メッセージおよび接続のプロトコル識別を可能にするソフトウェアメソッドを提供する。

【0205】

一実施形態によれば、ビルダが、サーバコード内で、第 1 のプロトコル A 412 および第 2 のプロトコル B 414 に対してプロトコルサポートを提供することができる。

【0206】

クライアントからサーバに向けられた接続要求メッセージが受信されると、要求を適切に処理することができるか否かに関して判定が行われる。プロトコル識別プロセスは、例えば、システムが要求プロトコルをサポートしていないこと、システムが、判断を行うのに十分なデータを有していないこと、またはそのような判定を行うために追加のデータが必要であることの標示を返すことができる。

【0207】

図25は、一実施形態による、接続のプロトコルを識別するためのサポートをさらに示す。

【0208】

図25に示すように、一実施形態によれば、拡張可能なサービスプロバイダインターフェース(SPI)402は、例えば、第1のプロトコルAに関連付けられた第1の接続プロバイダ404、第2のプロトコルBに関連付けられた第2の接続プロバイダ406、および、この例では、サーバによって現在サポートされていない可能性のある第3のまたは他のプロトコルNに関連付けられた第3の接続プロバイダ408を有することで、要求メッセージおよび接続のためのプロトコルを識別することができる。

10

【0209】

図26は、一実施形態による、接続のプロトコルを識別するためのサポートをさらに示す。

【0210】

図26に示すように、一実施形態によれば、第1のプロトコルA372を使用する接続のための要求A370、および第2のプロトコルB382を使用する接続のための要求B380を、適切なサーバコードによって処理することができる。

20

【0211】

図27は、一実施形態による、接続のプロトコルを識別するためのサポートをさらに示す。

【0212】

図27に示すように、一実施形態によれば、未知の他のプロトコルX412を使用する接続のための要求X410が、サーバによってサポートされていないと判定され、SPIは、接続要求を受け入れ、それを何らかの方法で処理するか、または接続要求を受け入れないかを判定することができる。

30

【0213】

一実施形態によれば、システムは、拡張可能なメカニズムを使用して、着信接続を正しく処理するためにそのタイプを識別することができる。SPIは、着信接続の最初のバイトを使用して、それがいかなる種類の接続であるか(HTTP/1、事前知識のあるHTTP/2など)を識別することができる。このメカニズムは、(各プロトコルハンドラで構成可能な)最初のある量のバイトを使用して識別を試みる。識別ができない場合、接続は受け入れられない。

【0214】

サービスプロバイダインターフェース例

一実施形態によれば、HTTP/1.1接続プロバイダを有するSPI実装は、最初のCRLFシーケンスまでバイトを使用し、それをパースしようとし、そのシーケンスは、有効なHTTP/1.1プロログ(例えばGET/path HTTP/1.1)を形成する場合、HTTP/1.1要求として受け入れられる。

40

【0215】

一実施形態によれば、HTTP/2.2接続プロバイダを有するSPI実装は、仕様によって定義されたコネクションプリフェイスを形成する一定数のバイトを使用する。

【0216】

このアプローチは拡張可能であり、他の接続プロバイダがこのメカニズムにプラグインして、他のプロトコルのサポートを提供することができる。

【0217】

50

一実施形態によれば、SPIおよびAPIは、サーバ接続プロバイダ（ServerConnectionProvider）として提供することができ、例えば以下のように、さまざまなインターフェース、クラス、またはメソッドを提供することができる。

【0218】

メソッド `int bytesToIdentifyConnection()` – 必要なバイトの数（「未知（unknown）」を識別するために0である場合もある）。

【0219】

メソッド `Support supports(BufferData data)` – `BufferData`は接続の初期バイトの抽象化である。`support`は、これがこのプロバイダにとって有効な要求であるか否か、識別するためにさらなるデータが必要であるか否か、またはこれがこのプロバイダにとって無効な要求であるか否かを定義する。

【0220】

メソッド `ServerConnection connection(ConnectionContext)` – このプロバイダが受け入れる接続に対するサーバ接続を作成し、`context`は、リーダおよびライタへのアクセスを提供する。

【0221】

TLSを使用するときのプロトコルネゴシエーションで使用するメソッド `SupportedApplicationProtocols()`。

【0222】

一実施形態によれば、例えば以下のように、サーバ接続プロバイダのための `java.util.ServiceLoader` プロバイダインターフェースを提供することができ、接続プロバイダに、要求バイトを分析し、これが受け入れることができる接続であるか否かを判断する機会が与えられる。

```
public interface ServerConnectionProvider {
```

```
    /**
```

```
     * この接続を識別するために必要なバイトの数。
```

```
     * @return 必要なバイトの数、これが固定値でない場合は0を返す
```

```
     */
```

```
    int bytesToIdentifyConnection();
```

```
    /**
```

```
     * このプロバイダは現在のサーバ接続をサポートしているか。
```

```
     * 同じバッファが{ServerConnection#handle()}に送られる
```

```
     * @param （少なくとも{bytesToIdentifyConnection()}の利用可能なバイトがある）
```

```
     * データバイト
```

```
     * @return サポート応答
```

```
     */
```

```
    Support supports(BufferData data);
```

```
    /**
```

```
     * 例えばALPNネゴシエーションのために使用される、
```

```
     * このプロバイダによってサポートされるアプリケーションプロトコル。
```

```
     * @return サポートされるプロトコルのセット
```

```
     */
```

```
    Set<String> supportedApplicationProtocols();
```

```
    /**
```

```
     * 新たな接続を作成する。
```

```
     * すべてのメソッドが、単一の仮想スレッドから呼び出される。
```

```
     * @param ctx データライタ、データリーダおよび他の有用な情報に
```

```
     * アクセスすることができる接続コンテキスト
```

```
     * @return 新たなサーバ接続
```

10

20

30

40

50

```

    */
    ServerConnection connection(ConnectionContext ctx);
    /**
     * このプロバイダによるサポート。
     */
    enum Support {
        /**
         * はい、これは、このプロバイダが処理することができる接続である。
         */
        SUPPORTED,
        /**
         * いいえ、この接続は、このプロバイダと互換性がない。
         */
        UNSUPPORTED,
        /**
         * 判定するのに十分なバイトがないため、このプロバイダに
         * 再度より多くのバイトを提供するように要求すること。
         */
        UNKNOWN
    }
}

```

10

20

【0223】

一実施形態によれば、サーバ接続用の S P I (ServerConnection) を提供することができ、この S P I は、ServerConnectionProviderによって作成することができ例えばサーバ接続の抽象化として、接続がクローズされるまでブロックすることが予期される単一のメソッド handle() を有し、例えば以下のように、ソケット接続を処理するために任意のプロバイダによって使用することができる。

```

public interface ServerConnection {
    /**
     * 接続の処理を開始する。 {ServerConnectionProvider#connection(webse
     rver.ConnectionContext)} を介してデータが提供される。
     * @throws 任意の待機状態を中断し、この接続を終了するための InterruptedEx
     ception
     */
    void handle() throws InterruptedException;
}

```

30

【0224】

一実施形態によれば、以下のように、H T T P / 1 サーバ接続プロバイダ (Http1ConnectionProvider) は、例えば H T T P / 1.1 サーバ接続プロバイダのための (java.util.ServiceLoader) プロバイダ実装として提供することができる。

```

public class Http1ConnectionProvider implements ServerConnectionPro
vider {
    private static final String PROTOCOL = " HTTP/1.1\r";
    private final int maxPrologueLength;
    private final int maxHeadersSize;
    private final boolean validateHeaders;
    private final boolean validatePath;
    private final Map<String, Http1UpgradeProvider> upgradeProvider
Map;
    /**

```

40

50

```

* デフォルト構成を有する新たなインスタンスを作成する。
* @deprecated {java.util.ServiceLoader}によってのみ使用されるべき
*/
@Deprecated
public Http1ConnectionProvider() {
    this(builder());
}
private Http1ConnectionProvider(Builder builder) {
    this.maxPrologueLength = builder.maxPrologueLength;
    this.maxHeadersSize = builder.maxHeaderSize;
    this.validateHeaders = builder.validateHeaders;
    this.validatePath = builder.validatePath;
    this.upgradeProviderMap = builder.upgradeProviders();
}
/**
* このプロバイダをセットアップすべきビルダ。
* @return 新たなビルダ
*/
public static Builder builder() {
    return new Builder();
}
@Override
public int bytesToIdentifyConnection() {
    // 要求は、\r\nで終わる行で開始しなければならない、および
    // 長さは分からない
    return 0;
}
@Override
public Support supports(BufferData request) {
    // 最初の\r\nを見つけ、\rが先行する場合、我々のものか試す
    // そうでない場合、サポートされていない
    /**
    > GET /loom/slow HTTP/1.1
    > Host: localhost:8080
    > User-Agent: curl/7.54.0
    > Accept: */*
    */
    int lf = request.indexOf(Bytes.LF_BYTE);
    if (lf == -1) {
        // 最大プロローグ長に達した場合、これをHTTP/1.1であるとみな
        // すだけであり、そのため、適切なエラーを送信することができる。つまり、maxPrologueLengthは、接続を識別するためにいかなるプロトコル要件よりも常に高くなければならない（例えば、これはフォールバックプロトコルである）
        return (request.available() <= maxPrologueLength) ?
            Support.SUPPORTED : Support.UNSUPPORTED;
    } else {
        return request.readString(lf).endsWith(PROTOCOL) ?
            Support.SUPPORTED : Support.UNSUPPORTED;
    }
}
}

```

```

@Override
public Set <String> supportedApplicationProtocols() {
    return Set.of("http/1.1");
}
@Override
public ServerConnection connection(ConnectionContext ctx) {
    return new Http1Connection(ctx, maxPrologueLength,
        maxHeadersSize, validateHeaders, validatePath, upgradeProv
iderMap);
}
/**
 * {io.webserver.http1.Http1ConnectionProvider}のためのFluent APIビルダ。
 */
public static class Builder implements common.Builder <Builder,
    Http1ConnectionProvider> {
    private final HelidonServiceLoader.Builder <Http1UpgradeProvider
>
        upgradeProviders =
            HelidonServiceLoader.builder(ServiceLoader.load(Http1UpgradeP
rovider.class)
    );
    private int maxPrologueLength = DEFAULT_MAX_PROLOGUE_LEN
GTH;
    private int maxHeaderSize = DEFAULT_MAX_HEADERS_SIZE;
    private boolean validateHeaders = DEFAULT_VALIDATE_HEADERS;
    private boolean validatePath = DEFAULT_VALIDATE_PATH;
    private Builder() {
    }
    @Override
    public Http1ConnectionProvider build() {
        return new Http1ConnectionProvider(this);
    }
    /**
     * 受信したHTTPプロローグ(GET /path HTTP/1.1)の最大サイズ。
     * @param maxPrologueLength バイトでの最大サイズ
     * @return 更新されたビルダ
     */
    public Builder maxPrologueLength(int maxPrologueLength) {
        this.maxPrologueLength = maxPrologueLength;
    }
    /**
     * バイトでの受信ヘッダの最大サイズ。
     * @param maxHeadersSize 最大ヘッダサイズ
     * @return 更新されたビルダ
     */
    public Builder maxHeadersSize(int maxHeadersSize) {
        this.maxHeaderSize = maxHeadersSize;
        return this;
    }

```

```

    }
    /**
     * ヘッダを検証すべきか否か。
     * falseに設定されている場合、いかなる値も受け入れられ、
     * そうでない場合は、ヘッダを検証し、既知のヘッダはフォーマットによって
     * 検証される（コンテンツ長は、プロトコル処理の一部であるため常に検証さ
れる
     * （他のヘッダは、機能がそれらを使用する場合に検証される可能性がある）
    ）。
```

10

```

    *
    * @param validateHeaders ヘッダを検証すべきか否か
    * @return 更新されたビルダ
    */
    public Builder validateHeaders(boolean validateHeaders) {
        this.validateHeaders = validateHeaders;
        return this;
    }
```

```

    /**
     * falseに設定されている場合、いかなるパスも受け入れられる
     * （不正文字を含んでいても）
     * @param validatePath パスを検証すべきか否か
     * @return 更新されたビルダ
    */
```

20

```

    public Builder validatePath(boolean validatePath) {
        this.validatePath = validatePath;
        return this;
    }
```

```

    /**
     * 構成されたアップグレードプロバイダを追加する。これにより、
     * サービスローダを通して発見されたインスタンス（存在する場合）が
     * 置き換えられる。
     * @param provider プロバイダを追加する
     * @return 更新されたビルダ
    */
```

30

```

    public Builder addUpgradeProvider(Http1UpgradeProvider provider) {
        upgradeProviders.addService(provider);
        return this;
    }
}
```

40

【0225】

一実施形態によれば、HTTP／2サーバ接続プロバイダ（Http2ConnectionProvider）を、例えば以下のように、「事前知識」要求をサポートするために提供することができる。

```

/**
 * HTTP2サーバ接続プロバイダのための{java.util.ServiceLoader}プロバイダ
実装
*/
```

```

public class Http2ConnectionProvider implements ServerConnectionPro
```

50

```

vider {
    @Override
    public int bytesToIdentifyConnection() {
        return PREFACE_LENGTH;
    }
    @Override
    public Support supports(BufferData request) {
        byte[] prefaceBytes = new byte[PREFACE_LENGTH];
        request.read(prefaceBytes, 0, PREFACE_LENGTH);
        // これで、プロトコルハンドラにこのプロトコルを識別するように
        // 求めることができる
        if (isPreface(prefaceBytes)) {
            // これはHTTP 2 事前知識である
            return Support.SUPPORTED;
        }
        return Support.UNSUPPORTED;
    }
    @Override
    public Set<String> supportedApplicationProtocols() {
        return Set.of("h2");
    }
    @Override
    public ServerConnection connection(ConnectionContext ctx) {
        Http2Connection result = new Http2Connection(ctx);
        result.expectPreface();
        return result;
    }
}

```

【0226】
 上記の例は、例示の目的で提供するものであり、さまざまな実施形態によれば、インターフェース、クラス、またはメソッドの他の例を提供することができる。

【0227】
 図28は、一実施形態によれば、接続のプロトコルを識別するための、マイクロサービス環境で使用される方法を示す。

【0228】
 図28に示すように、一実施形態によれば、ステップ420において、1つまたは複数のプロセッサおよびメモリを含むコンピュータにおいて、クライアントアプリケーションおよびサーバアプリケーションがマイクロサービス環境内で通信することを可能にする、（例えば、マイクロサービスライブラリによって提供されるような）マイクロサービス環境が提供される。

【0229】
 ステップ422において、システムは、例えばHTTPの要求ルーティングで 사용되는ように、サーバコードによってサポートされる複数のプロトコルをサポートするために拡張可能な接続プロバイダサービスプロバイダインタフェース（SPI）を含む、接続プロトコル識別コンポーネントまたは機能性を提供する。

【0230】
 ステップ424において、接続プロバイダSPIは複数の接続プロバイダタイプを提供し、接続プロバイダタイプの各々は、サポートされるプロトコルに関連付けられる。

【0231】
 ステップ426において、着信接続を介してサーバに向けられた要求を受信すると、シ

システムは、着信要求の初期バイトを使用して、接続タイプ（例えば、HTTP / 1、HTTP / 2）を識別し、接続要求を処理するか、または他の方法で、接続要求がサポートされていないとして拒否することができる。

【0232】

インターフェース例

一実施形態によれば、説明の目的で、接続のプロトコルを識別するためのインターフェース例を以下に提供する。他の実施形態によれば、他のタイプのインターフェースを提供することができる。

【0233】

インターフェース `ServerConnectionProvider`

一実施形態によれば、`ServerConnectionProvider` インターフェースは、サーバ接続プロバイダ用のインターフェースを提供するために使用することができ、接続プロバイダは、要求バイトを分析し、これが受け入れることができる接続であるか否かを判断する機会を与えられ、例えば以下のような、さまざまなメソッドを含むことができる。

【0234】

メソッド `int bytesToIdentifyConnection()`。この接続を識別するために必要なバイトの数。一実施形態では、このメソッドは必要なバイト数を返し、これが固定値でない場合は0を返す。

【0235】

メソッド `ServerConnectionProvider.Support supports (common.buffer.BufferData data)`。このプロバイダが現在のサーバ接続をサポートしているか。同じバッファが `ServerConnection.handle()` に送られる。提供することができるパラメータとしては、例えば、データ（少なくとも `bytesToIdentifyConnection()` の利用可能なバイトを含む）バイトが挙げられる。一実施形態では、このメソッドはサポート応答を返す。

【0236】

メソッド `Set<String> supportedApplicationProtocols()`。例えばALPNネゴシエーションに使用される、このプロバイダによってサポートされるアプリケーションプロトコル。一実施形態では、このメソッドは、サポートされるプロトコルのセットを返す。

【0237】

メソッド `ServerConnection connection(ConnectionContext ctx)`。新たな接続を作成する。すべてのメソッドは、単一の仮想スレッドから呼び出される。提供することができるパラメータとしては、例えば、ctx-データライタ、データリーダ、および他の有用な情報にアクセスすることができる接続コンテキストが挙げられる。一実施形態では、このメソッドは、新たなサーバ接続を返す。

【0238】

さまざまな実施形態によれば、本開示の態様を、以下の番号付けされた条項において説明する。

【0239】

1. マイクロサービスまたは他のコンピューティング環境で使用されるシステムであって、1つまたは複数のプロセッサを含み、ソフトウェアアプリケーションまたはサービスとともに使用されるマイクロサービスまたは他のコンピューティング環境へのアクセスを提供するコンピュータを備え、システムは、ヘッダ列挙を介して、メッセージヘッダがオブジェクトとして扱われることを可能にする抽象化を提供し、ヘッダ列挙は、複数の通信プロトコルをサポートし、クライアントおよびサーバが、いくつかのサポートされるプロトコルのうちの任意のものを使用して要求メッセージおよび応答メッセージを通信することを可能にするアプリケーションプログラムインターフェースを介してアクセス可能である、

【0240】

2. 条項1に記載のシステムであって、既知のヘッダタイプを有する要求メッセージに遭遇したとき、システムは、利用可能であれば、上記列挙からインデックス付きの値を取得し、関連付けられたデータを、要求が向けられているプロセスに直接提供し、および／または、後の使用のためにヘッダの値をキャッシュすることができる。

【0241】

3. マイクロサービスまたは他のコンピューティング環境で使用される方法であって、ソフトウェアアプリケーションまたはサービスとともに使用されるマイクロサービスまたは他のコンピューティング環境を提供することと、ヘッダ列挙によって、メッセージヘッダが、複数の通信プロトコルをサポートし、クライアントおよびサーバが、サポートされているプロトコルのうちの任意のものを使用して要求／応答メッセージを通信することを可能にするアプリケーションプログラムインターフェースを介してアクセス可能な、オブジェクトとして扱われることを可能にする抽象化を提供することを含む、方法。

10

【0242】

4. 条項3に記載の方法であって、既知のヘッダタイプを有する要求メッセージに遭遇したとき、システムは、利用可能であれば、上記列挙からインデックス付きの値を取得し、関連付けられたデータを、要求が向けられているプロセスに直接提供し、および／または、後の使用のためにヘッダの値をキャッシュすることができる。

【0243】

5. マイクロサービスまたは他のコンピューティング環境で使用されるシステムであって、1つまたは複数のプロセッサを含み、ソフトウェアアプリケーションまたはサービスとともに使用されるマイクロサービスまたは他のコンピューティング環境へのアクセスを提供するコンピュータを備え、ソフトウェアアプリケーションまたはサービスのテスト中に、クライアント接続が、ネットワーク通信プロトコルの使用をエミュレートし、サーバ接続に逆関数を公開するインメモリ実装に置き換えられ、クライアント接続のデータライタに書き込まれたデータが、サーバ接続においてデータリーダを通して利用可能になること、および、サーバ接続のデータライタに書き込まれたデータが、クライアント接続においてデータリーダを通して利用可能になることを含む、システム。

20

【0244】

6. 条項5に記載のシステムであって、インメモリ実装は、メモリ内のブロッキングキューを使用し、実際のソケットを使用せずにネットワーク通信プロトコルの使用をエミュレートし、テストは完了しており、クライアントおよび／またはサーバコードは、実際のソケットに対して実行することができる。

30

【0245】

7. マイクロサービスまたは他のコンピューティング環境で使用される方法であって、ソフトウェアアプリケーションまたはサービスとともに使用されるマイクロサービスまたは他のコンピューティング環境を提供することと、ソフトウェアアプリケーションまたはサービスのテスト中に、クライアント接続を、ネットワーク通信プロトコルの使用をエミュレートし、サーバ接続に逆関数を公開するインメモリ実装に置き換えることとを含み、クライアント接続のデータライタに書き込まれたデータが、サーバ接続においてデータリーダを通して利用可能になること、および、サーバ接続のデータライタに書き込まれたデータが、クライアント接続においてデータリーダを通して利用可能になることを含む、方法。

40

【0246】

8. 条項7に記載の方法であって、インメモリ実装は、メモリ内のブロッキングキューを使用し、実際のソケットを使用せずにネットワーク通信プロトコルの使用をエミュレートし、テストは完了しており、クライアントおよび／またはサーバコードは、実際のソケットに対して実行することができる。

【0247】

9. マイクロサービスまたは他のコンピューティング環境で使用されるシステムであって、1つまたは複数のプロセッサを含み、ソフトウェアアプリケーションまたはサービス

50

とともに使用されるマイクロサービスまたは他のコンピューティング環境へのアクセスを提供するコンピュータを備え、システムは、複数のネットワークプロトコルまたは通信プロトコルを介して要求に対応するように動作することができるサーバの構成を可能にし、サーバによってサポートされる各ネットワークプロトコルまたは通信プロトコルを、ネットワークプロトコルまたは通信プロトコル自体のルーティングと関連付けるルート抽象化が提供されること、および、ルート抽象化は、特定のネットワークプロトコルまたは通信プロトコルで使用する1つまたは複数の特定のルートを含むことができることを含む、システム。

【0248】

10. 条項9に記載のシステムであって、特定のプロトコルを介して受信された要求メッセージに対して、システムは、要求に対する適切なルートを選択し、ルート抽象化に従って要求を処理することができる。

10

【0249】

11. マイクロサービスまたは他のコンピューティング環境で使用される方法であって、ソフトウェアアプリケーションまたはサービスとともに使用されるマイクロサービスまたは他のコンピューティング環境を提供することを含み、システムは、複数のネットワークプロトコルまたは通信プロトコルを介して要求に対応するように動作することができるサーバの構成を可能にし、サーバによってサポートされる各ネットワークプロトコルまたは通信プロトコルを、ネットワークプロトコルまたは通信プロトコル自体のルーティングと関連付けるルート抽象化が提供されること、および、ルート抽象化は、特定のネットワークプロトコルまたは通信プロトコルで使用する1つまたは複数の特定のルートを含むことができることを含む、方法。

20

【0250】

12. 条項11に記載の方法であって、特定のプロトコルを介して受信された要求メッセージに対して、システムは、要求に対する適切なルートを選択し、ルート抽象化に従って要求を処理することができる。

【0251】

13. マイクロサービスまたは他のコンピューティング環境で使用されるシステムであって、1つまたは複数のプロセッサを含み、ソフトウェアアプリケーションまたはサービスとともに使用されるマイクロサービスまたは他のコンピューティング環境へのアクセスを提供するコンピュータと、要求メッセージおよびクライアント-サーバ接続に関連付けられたネットワーク通信プロトコルの識別を可能にするソフトウェアメソッドを提供する拡張可能なサービスプロバイダインターフェースとを備え、ネットワーク通信プロトコルの識別は、クライアント-サーバ接続の抽象化、ルータの抽象化、または単一のソケット上での複数のネットワーク通信プロトコルプロトコルの使用に対するサポートのうちの1つまたは複数を提供するために使用される、システム。

30

【0252】

14. 条項13に記載のシステムであって、接続要求メッセージがクライアントから受信され、サーバに向けられたとき、要求を、提供されたネットワーク通信プロトコルを使用して処理することができるか否かに関して、判定が行われ、プロトコル識別プロセスが、システムが要求ネットワーク通信プロトコルをサポートしていないこと、またはシステムが現在、判断を行うのに十分なデータを有していないこと、またはそのような判定を行うために追加のデータが必要であることの標示を返す。

40

【0253】

15. マイクロサービスまたは他のコンピューティング環境で使用される方法であって、ソフトウェアアプリケーションまたはサービスとともに使用されるマイクロサービスまたは他のコンピューティング環境を提供することと、要求メッセージおよびクライアント-サーバ接続に関連付けられたネットワーク通信プロトコルの識別を可能にするソフトウェアメソッドを提供する拡張可能なサービスプロバイダインターフェースを提供することを含み、ネットワーク通信プロトコルの識別は、クライアント-サーバ接続の抽象化、

50

ルータの抽象化、および単一のソケット上での複数のネットワーク通信プロトコルプロトコルの使用のうちの1つまたは複数を提供するために使用される、方法。

【0254】

16. 条項15に記載の方法であって、接続要求メッセージがクライアントから受信され、サーバに向けられたとき、要求を、提供されたネットワーク通信プロトコルを使用して処理することができるか否かに関して、判定が行われ、プロトコル識別プロセスが、システムが要求ネットワーク通信プロトコルをサポートしていないこと、またはシステムが現在、判断を行うのに十分なデータを有していないこと、またはそのような判定を行うために追加のデータが必要であることの標示を返す。

【0255】

17. 前述の番号付けされた条項のいずれかに記載のシステムまたは方法であって、システムまたは方法は、ウェブサーバの一部として、またはウェブサーバとともに使用されるように提供され、要求はHTTP要求である。

【0256】

18. 前述の番号付けされた条項のいずれかに記載のシステムまたは方法であって、システムまたは方法は、クラウドネイティブサービスをビルドするために使用可能な関連するライブラリおよび機能とともに仮想スレッドを使用する要求処理コンポーネントを含む。

【0257】

19. 前述の番号付けされた条項のいずれかによって記載されるシステムまたは方法であって、システムまたは方法は、1つまたは複数のクラウド、データベース、または他のシステムもしくはサービスへのアクセスを提供するクラウドコンピューティング環境内で提供される。

【0258】

さまざまな実施形態によれば、本明細書の教示は、本明細書の教示に従ってプログラムされた1つまたは複数のプロセッサ、メモリおよび／またはコンピュータ可読記憶媒体を含む、1つまたは複数のコンピュータ、コンピューティングデバイス、マシン、またはマイクロプロセッサを使用して実施することができる。ソフトウェアの技術分野における当業者には明らかとなるように、熟練したプログラマーは、本開示の教示に基づいて、適切なソフトウェアコーディングを容易に作成することができる。

【0259】

いくつかの実施形態において、本明細書の教示は、本教示のプロセスのうちの任意のものを実行するようにコンピュータをプログラムするために使用することができる命令が格納された、非一時的コンピュータ可読記憶媒体（複数の媒体）である、コンピュータプログラム製品を含むことができる。このような記憶媒体の例としては、限定されないが、ハードディスクドライブ、ハードディスク、ハードドライブ、固定ディスク、ROM、RAM、EPROM、EEPROM、DRAM、VRAM、フラッシュメモリデバイス、または命令および／またはデータの非一時的格納に適した他のタイプの記憶媒体またはデバイスを挙げることができる。

【0260】

前述の説明は、例示および説明を目的として提供されている。網羅的であるように、または保護範囲を開示した正確な形態に限定するようには意図されていない。さらなる変更および変形が当業者には明らかとなる。

【0261】

例えば、本明細書で説明したシステムおよび方法のさまざまな実施形態は、Helidonマイクロサービス環境での使用を示すが、さまざまな実施形態は、他のタイプのマイクロサービス環境または他のコンピューティング環境とともに使用することができる。

【0262】

実施形態は、本明細書における教示の原理およびそれらの実際的な適用を最もよく説明し、それにより、他の当業者が、さまざまな実施形態を、企図された特定の使用に適した

10

20

30

40

50

さまざまな変更とともに理解することができるようにするために、選択および記載されている。その範囲は、以下の請求項およびそれらの均等物により画定されると意図されている。

【図面】

【図 1】

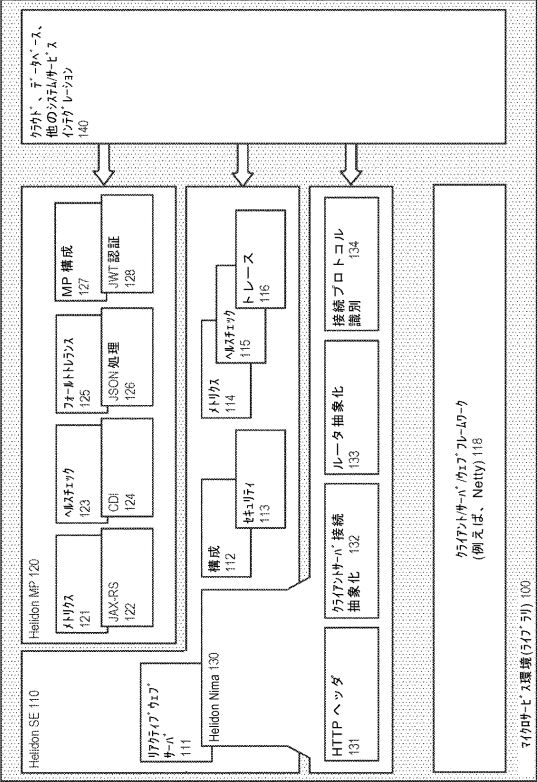


FIGURE 1

【図 2】

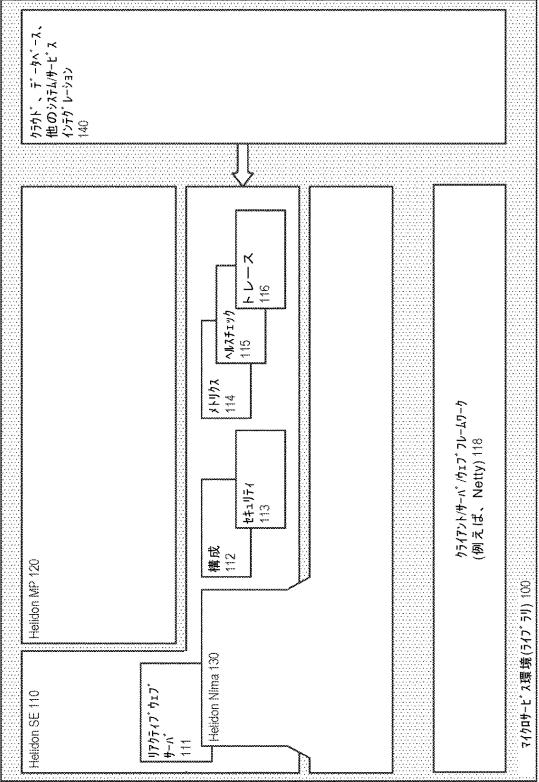


FIGURE 2

【図 3】

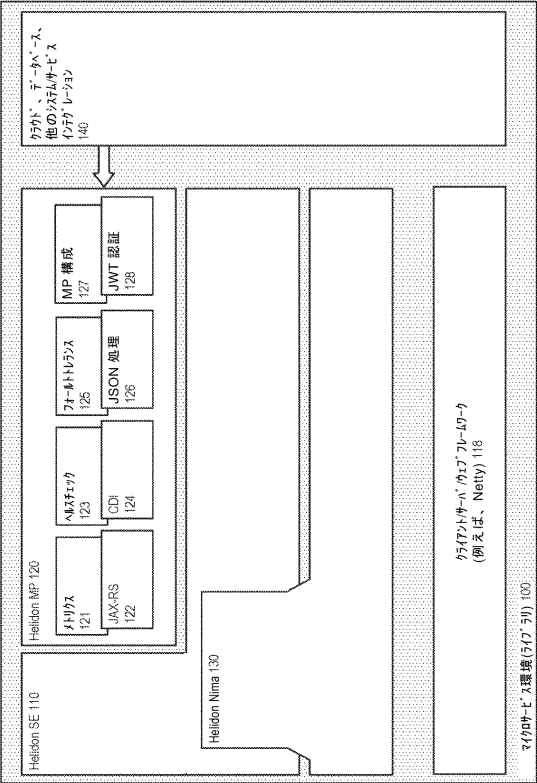


FIGURE 3

【図 4】

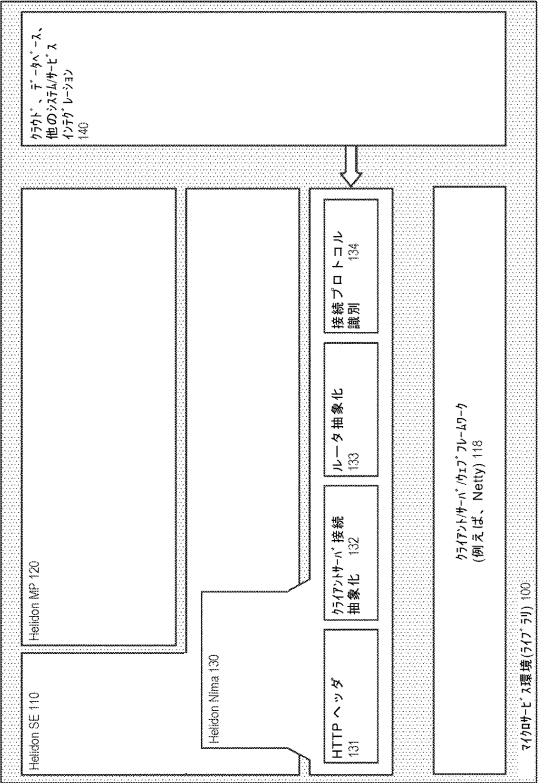
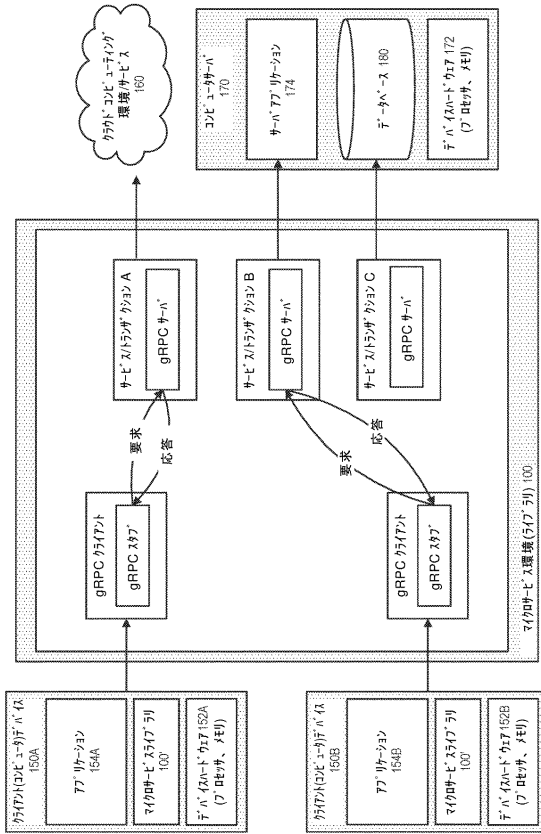


FIGURE 4

【図 5】



【図 6】

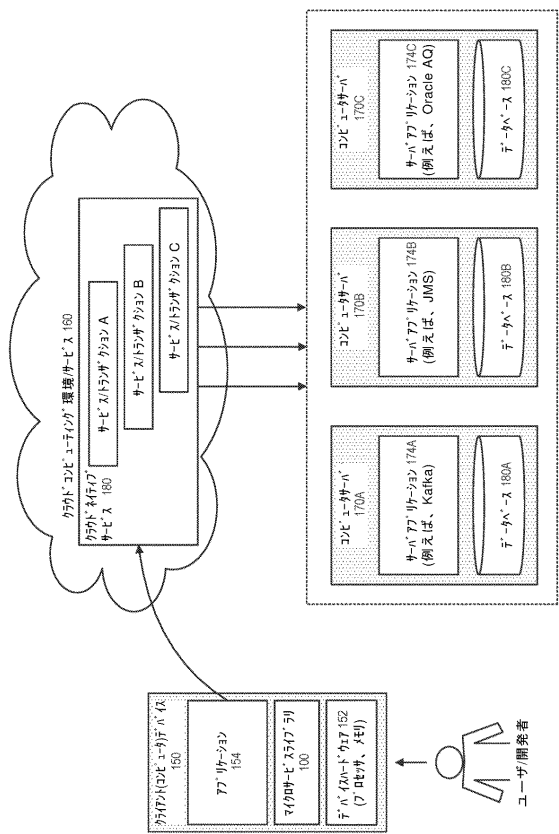


FIGURE 5

FIGURE 6

【図 7】

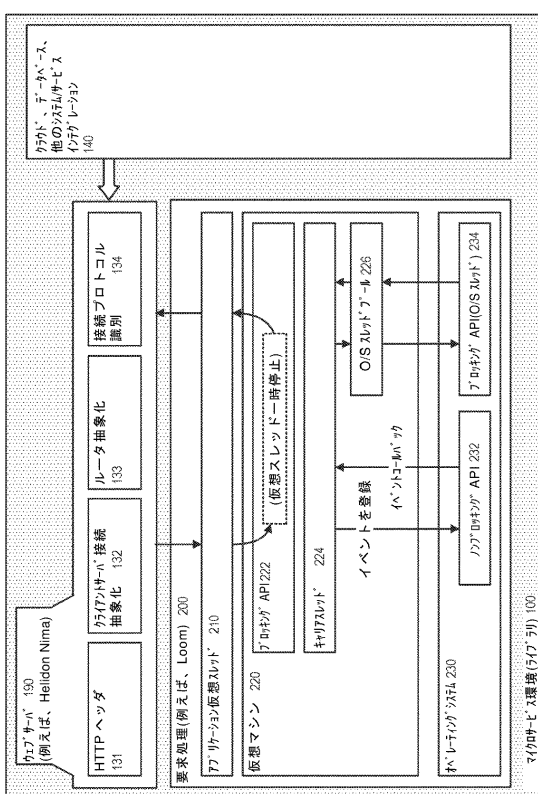


FIGURE 7

【図 8】

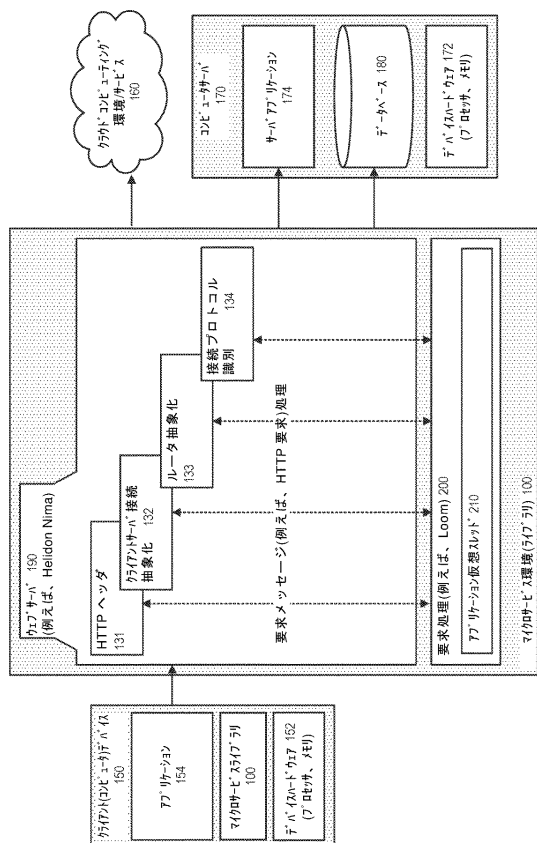


FIGURE 8

10

20

30

40

50

【図 9】

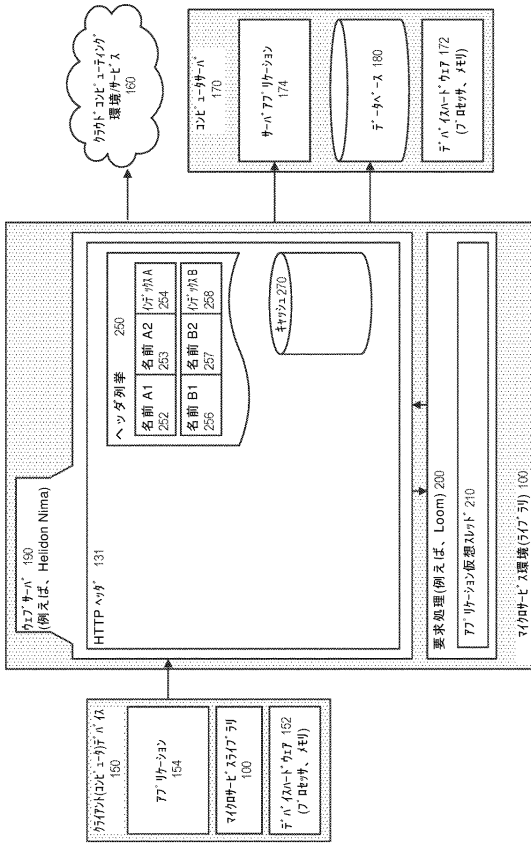


FIGURE 9

【図 10】

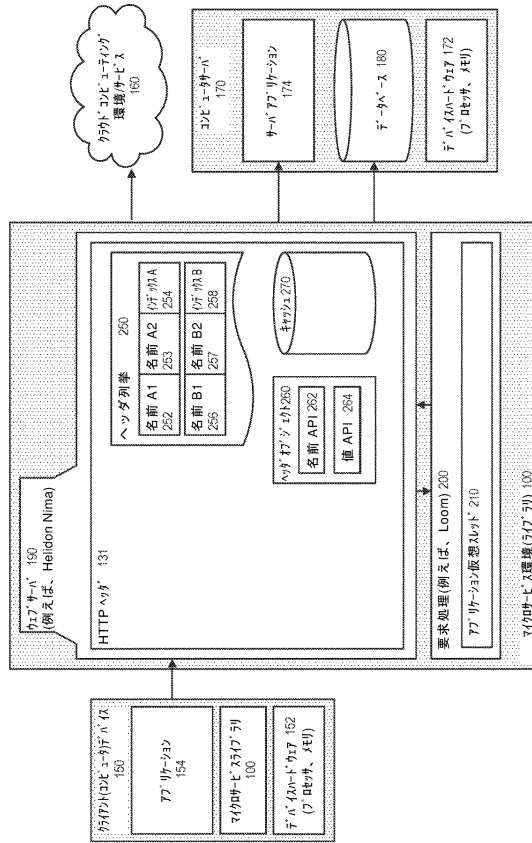


FIGURE 10

【図 11】

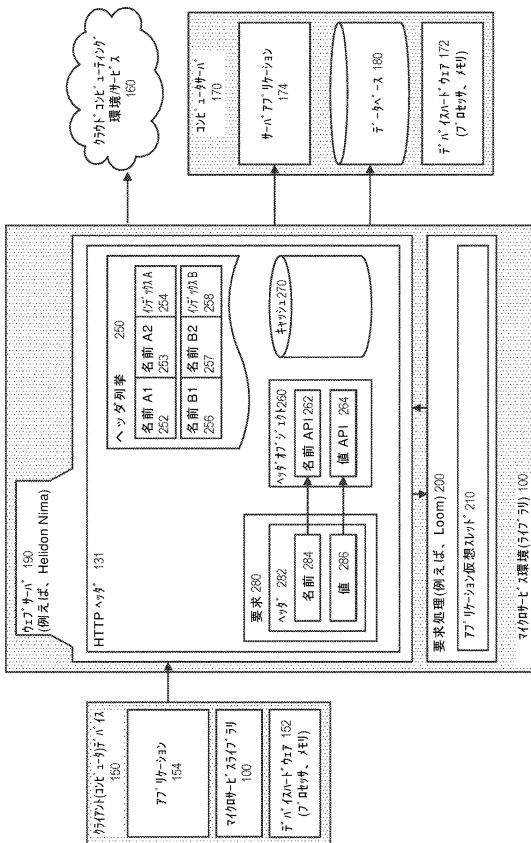


FIGURE 11

【図 12】

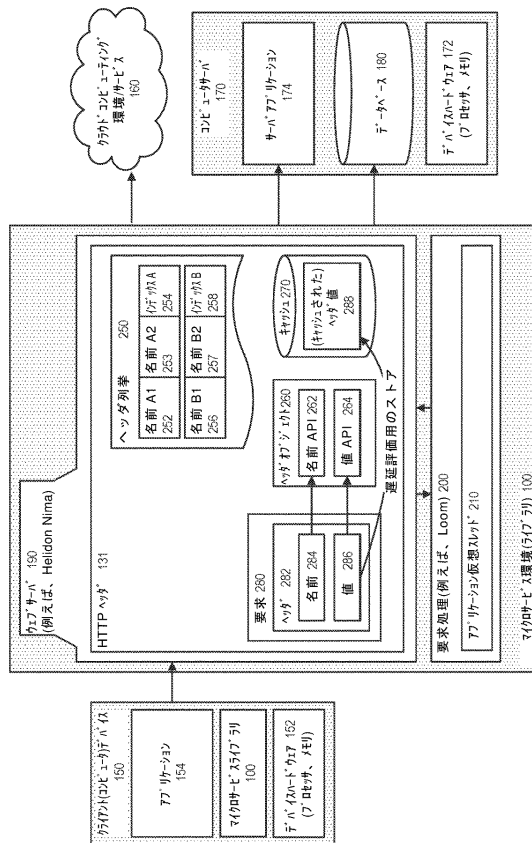


FIGURE 12

10

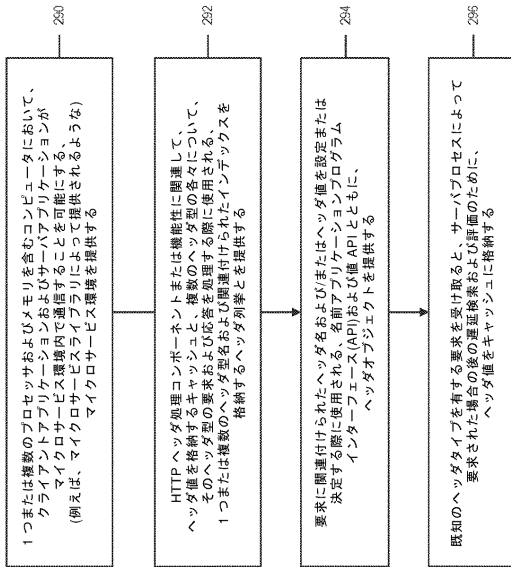
20

30

40

50

【図 13】



【図 14】

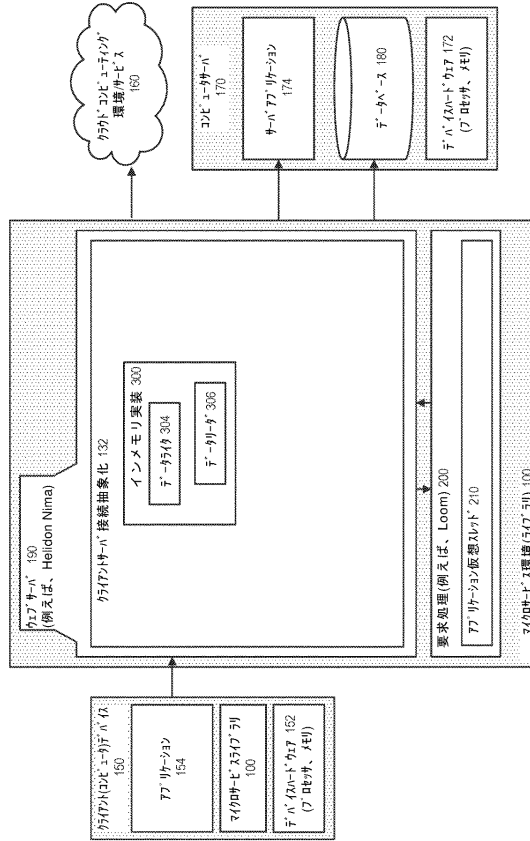


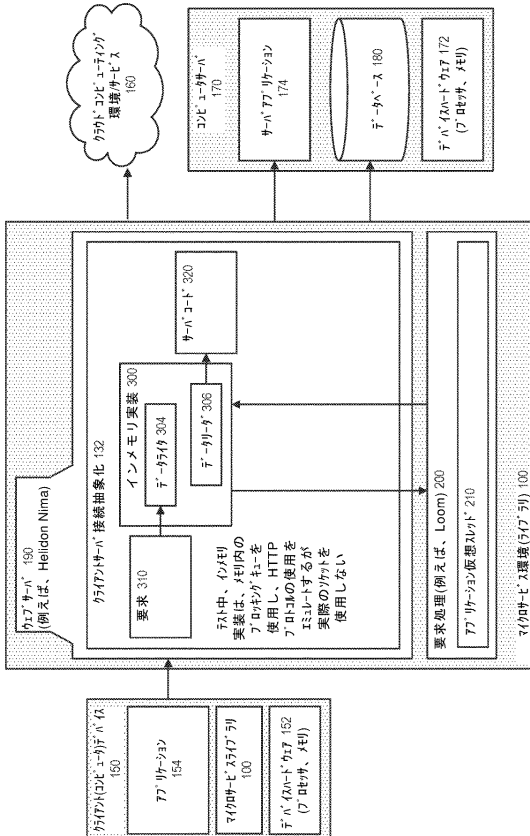
FIGURE 13

FIGURE 14

10

20

【図 15】



【図 16】

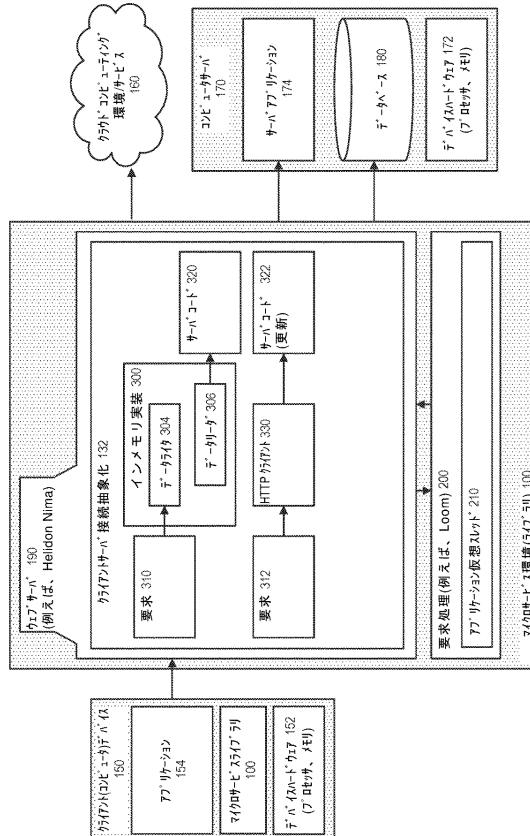


FIGURE 15

FIGURE 16

30

40

50

【図 17】

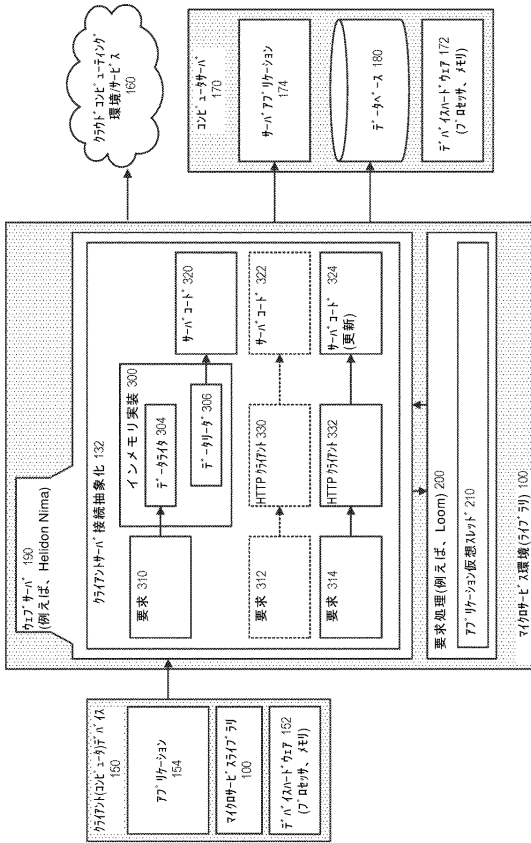


FIGURE 17

【図 18】

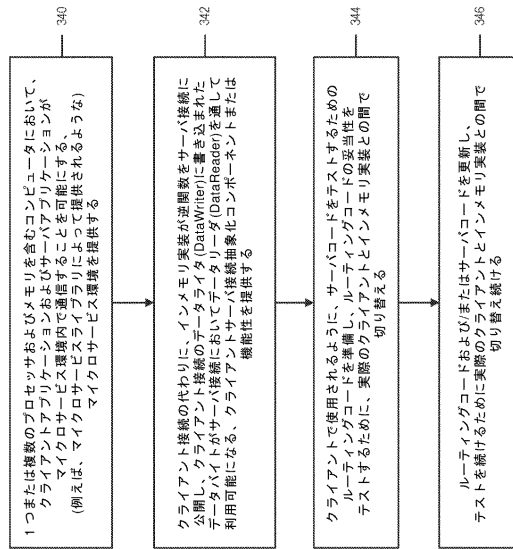


FIGURE 18

【図 19】

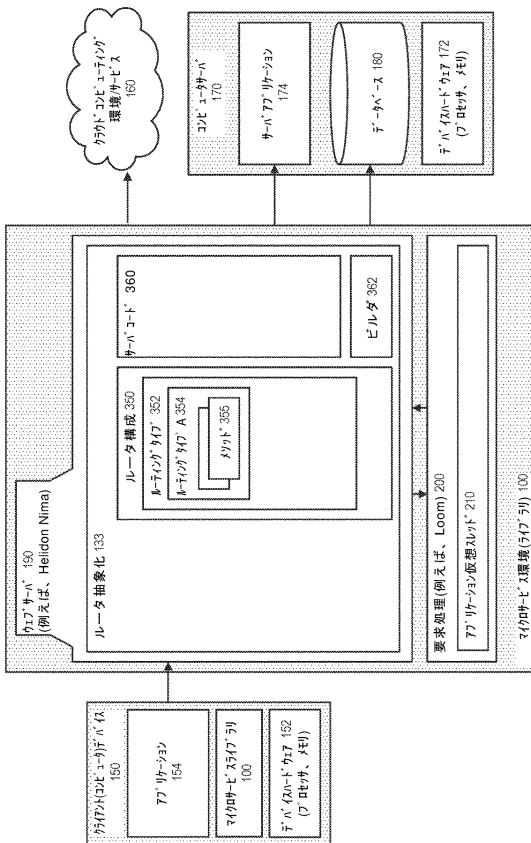


FIGURE 19

【図 20】

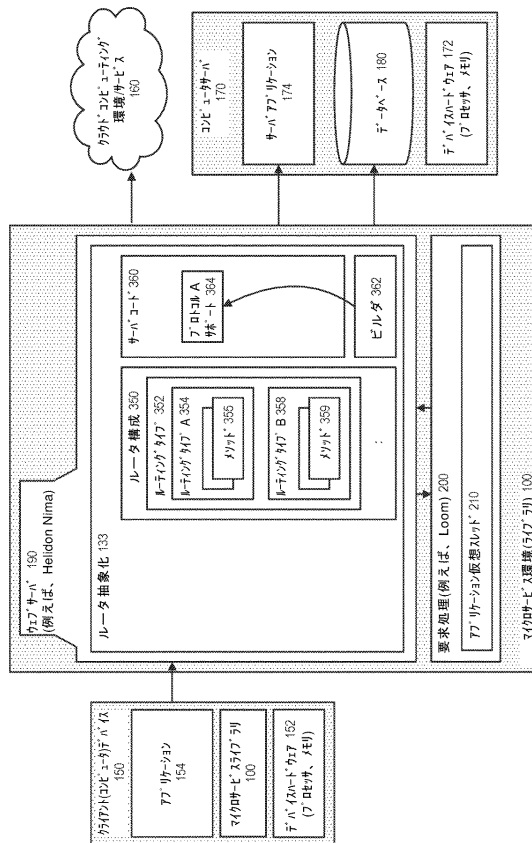


FIGURE 20

10

20

30

40

50

【図 25】

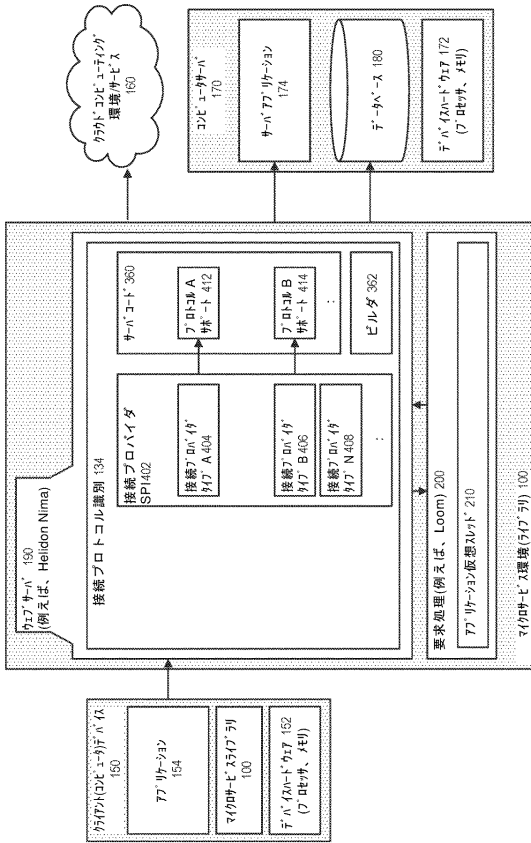


FIGURE 25

【図 26】

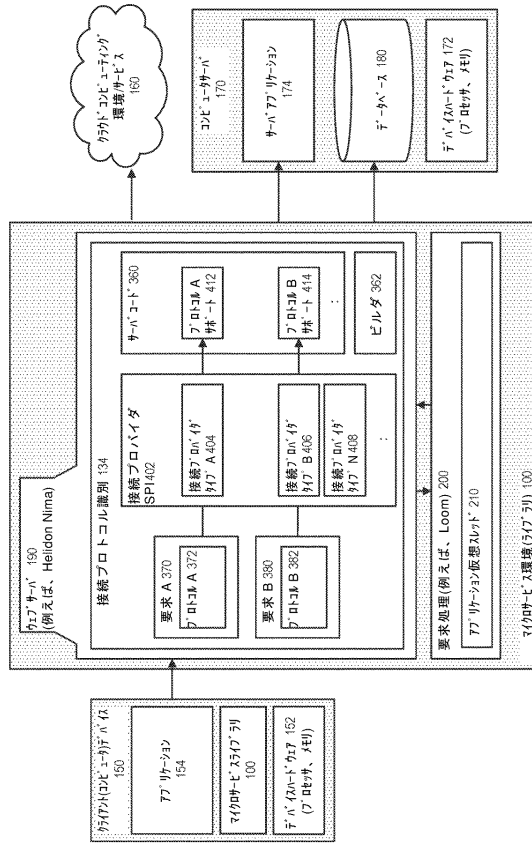


FIGURE 26

【図 27】

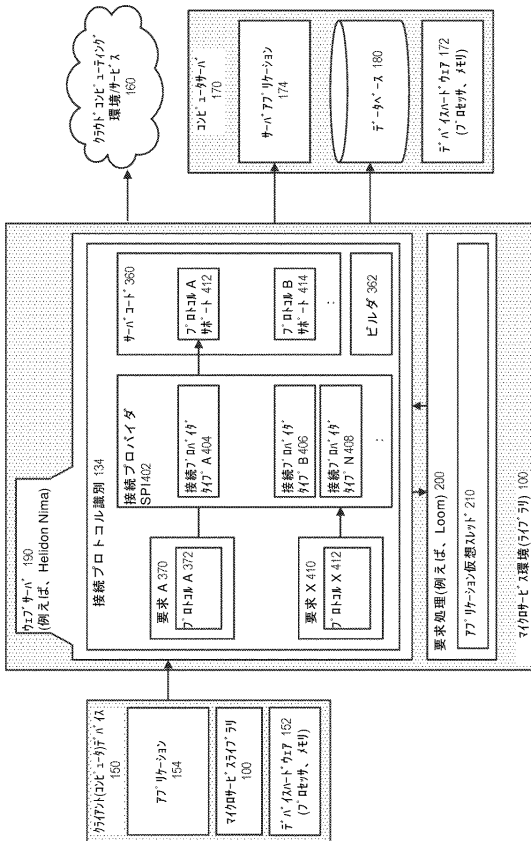


FIGURE 27

【図 28】

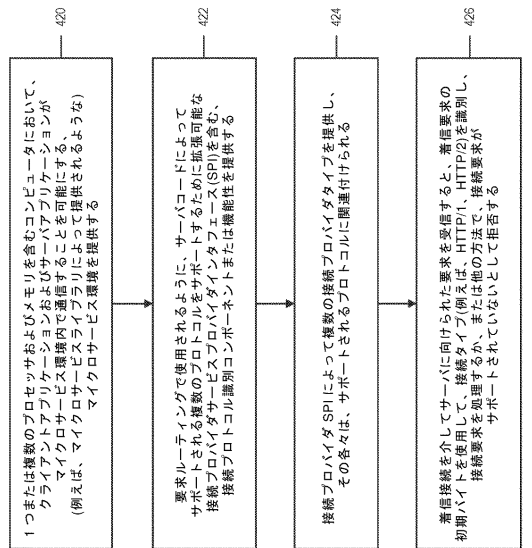


FIGURE 28

10

20

30

40

50

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2023/022765

A. CLASSIFICATION OF SUBJECT MATTER INV. H04L45/76 H04L67/02 H04L45/302 ADD.		
According to International Patent Classification (IPC) or to both national classification and IPC		
B. FIELDS SEARCHED		
Minimum documentation searched (classification system followed by classification symbols) H04L		
Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched		
Electronic data base consulted during the international search (name of data base and, where practicable, search terms used) EPO-Internal		
C. DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2006/268857 A1 (BESSIS THIERRY [US] ET AL) 30 November 2006 (2006-11-30)	1-3, 13-15, 25
Y	abstract paragraph [0011] - paragraph [0095]; figures 1-9	6-9, 18-21
A	US 10 331 418 B2 (PAYPAL INC [US]) 25 June 2019 (2019-06-25) the whole document	4, 5, 16, 17
Y	WO 2017/070545 A1 (INTERDIGITAL TECH CORP [US]) 27 April 2017 (2017-04-27) paragraph [0215] - paragraph [0270]	6, 7, 18, 19
Y	US 10 237 176 B2 (JUNIPER NETWORKS INC [US]) 19 March 2019 (2019-03-19) paragraph [0013] - paragraph [0097]	8, 9, 20, 21
☐ Further documents are listed in the continuation of Box C. ☒ See patent family annex.		
* Special categories of cited documents : "A" document defining the general state of the art which is not considered to be of particular relevance "E" earlier application or patent but published on or after the international filing date "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified) "O" document referring to an oral disclosure, use, exhibition or other means "P" document published prior to the international filing date but later than the priority date claimed "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art "&" document member of the same patent family		
Date of the actual completion of the international search 10 November 2023		Date of mailing of the international search report 20/11/2023
Name and mailing address of the ISA/ European Patent Office, P.B. 5618 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016		Authorized officer Manthey, Axel

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2023/022765**Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)**

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:
2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:
3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

10

20

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

see additional sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.
2. ☐ As all searchable claims could be searched without effort justifying an additional fees, this Authority did not invite payment of additional fees.
3. ☒ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:
1-9, 13-21, 25
4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims;; it is covered by claims Nos.:

30

40

Remark on Protest

- ☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.
- ☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.
- ☒ No protest accompanied the payment of additional search fees.

International Application No. PCT/US2023/022765

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. claims: 1-3, 13-15(completely); 25(partially)

A system, method and a non-transitory computer readable storage medium for protocol header processing

2. claims: 4, 5, 16, 17(completely); 25(partially)

A system, method and a non-transitory computer readable storage medium for providing a software verification tool

3. claims: 6, 7, 18, 19(completely); 25(partially)

A system, method and a non-transitory computer readable storage medium for providing route abstraction

4. claims: 8, 9, 20, 21(completely); 25(partially)

A system, method and a non-transitory computer readable storage medium for enabling third party services

5. claims: 10, 22(completely); 25(partially)

A system, method and a non-transitory computer readable storage medium for providing web services

6. claims: 11, 23(completely); 25(partially)

A system, method and a non-transitory computer readable storage medium for microservices

7. claims: 12, 24(completely); 25(partially)

A system, method and a non-transitory computer readable storage medium for virtualization

10

20

30

40

50

INTERNATIONAL SEARCH REPORT				International application No PCT/US2023/022765	
Information on patent family members					
Patent document cited in search report		Publication date	Patent family member(s)		Publication date
US 2006268857	A1	30-11-2006	NONE		

US 10331418	B2	25-06-2019	US 2017161028 A1		08-06-2017
			US 2019317736 A1		17-10-2019

WO 2017070545	A1	27-04-2017	NONE		

US 10237176	B2	19-03-2019	CN 107566440 A		09-01-2018
			CN 112039748 A		04-12-2020
			EP 3264691 A1		03-01-2018
			US 2018006935 A1		04-01-2018
			US 2019190824 A1		20-06-2019

10

20

30

40

50

フロントページの続き

(32)優先日 令和5年5月17日(2023.5.17)

(33)優先権主張国・地域又は機関
米国(US)

(31)優先権主張番号 18/198,762

(32)優先日 令和5年5月17日(2023.5.17)

(33)優先権主張国・地域又は機関
米国(US)

(31)優先権主張番号 18/198,764

(32)優先日 令和5年5月17日(2023.5.17)

(33)優先権主張国・地域又は機関
米国(US)

(31)優先権主張番号 18/198,765

(32)優先日 令和5年5月17日(2023.5.17)

(33)優先権主張国・地域又は機関
米国(US)

(81)指定国・地域 AP(BW,CV,GH,GM,KE,LR,LS,MW,MZ,NA,RW,SD,SL,ST,SZ,TZ,UG,ZM,ZW),EA(AM,AZ,BY,KG,KZ,RU,TJ,TM),EP(AL,AT,BE,BG,CH,CY,CZ,DE,DK,EE,ES,FI,FR,GB,GR,HR,HU,IE,IS,IT,LT,LU,LV,MC,ME,MK,MT,NL,NO,PL,PT,RO,RS,SE,SI,SK,SM,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,KM,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AO,AT,AU,AZ,BA,BB,BG,BH,BN,BR,BW,BY,BZ,CA,CH,CL,CN,CO,CR,CU,CV,CZ,DE,DJ,DK,DM,DO,DZ,EC,EE,EG,ES,FI,GB,GD,GE,GH,GM,GT,HN,HR,HU,ID,IL,IN,IQ,IR,IS,IT,JM,JO,JP,KE,KG,KH,KN,KP,KR,KW,KZ,LA,LC,LK,LR,LS,LU,LY,MA,MD,MG,MK,MN,MU,MW,MX,MY,MZ,NA,NG,NI,NO,NZ,OM,PA,PE,PG,PH,PL,PT,QA,RO,RS,RU,RW,SA,SC,SD,SE,SG,SK,SL,ST,SV,SY,TH,TJ,TM,TN,TR,TT,TZ,UA,UG,US,UZ,VC,VN,WS,ZA,ZM,ZW

(特許庁注：以下のものは登録商標)

1. J A V A