マルチテナント型アプリケーション開発の実装考慮点

Design and Implementation of Multitenant Services and Applications

中　村　誠　吾，田　中　　要

要　約　SaaS の提供者は，利用者により良いサービスを提供するために様々な工夫をしている．その中の一つにマルチテナントと呼ばれる考え方がある．

この考えを実現するシステムの実装方法のポイントを整理し，主にアプリケーション領域においてサービス品質保証とデータ領域の視点から実装方法をパターン化する．また，実際に日本ユニシスが提供しているサービスから Java を利用した RENANDI SaaS と .NET Framework を利用した SaaS アプリケーションを事例に取り上げ，顧客ごとにアプリケーション実行環境，データベースを分離する方式の具体的な実装方法を紹介する．本稿で述べる実装方法はアプリケーション構築のための方法であり，サービス提供開始後を考えた場合に，運用状況監視方法などの課題がある．

Abstract　SaaS providers have been making various efforts to improve their better services and to satisfy more customers. Their efforts have brought up some new ideas, one of which is multitenancy.

This paper describes how to design and implement multitenant services and applications aspect of the data area and the service level agreement. First, it explains key points to design the application architecture for implementing multitenancy and makes patterns of implementing multitenancy. Second, it introduces two types of services hosted by Nihon Unisys group, one of which is RENANDI SaaS implemented using Java and the other of which is SaaS Application implemented using .NET Framework. In these two services, all application executions of individual tenants are isolated by tenant each other and their databases are also isolated. Lastly, it illustrates their specific approaches of application implementation. In the future, we will have to approach the challenges of method for monitoring server resources after launching service provision.

1. は　じ　め　に

システムを所有するのではなく，サービスとして利用する，SaaS（Software as a Service）と呼ばれるサービスが注目を集めている．必要な時に，必要なだけ利用できることを特徴とし，バージョンアップの作業やシステムの運用が必要ないことから，利用者がシステムを所有することによって発生する無駄なコストを削減することに貢献している．

これらの特徴を顧客に提供するために，サービス提供者は様々な工夫をしている．その工夫の一つにマルチテナントと呼ばれる考え方がある．マルチテナントとは一つのハードウェア資源を複数の顧客が利用できるようにし，効率的に活用することである．この技術によってサービス提供者は顧客からサービス利用の申請があると，利用していない資源を柔軟に必要なサービスに割り当てることができ，顧客に必要な時に，必要なだけサービスを提供できるようになる．また，効率的な資源の活用や，アプリケーションの保守と運用作業の共通化が可能となるため，顧客にはより安価にサービスの提供が可能となる．

本稿ではサービスの開発で得られた経験から，マルチテナント技術を実装するために考慮すべき点と，実装方法の違いによる特徴をまとめている．さらに日本ユニシスが Java と .NET Framework を利用して構築・提供している二つのサービスを例に，具体的なマルチテナント型アプリケーションの実装方法を示す．

2. マルチテナント技術

マルチテナント技術とは，物理的に分離されていないコンピューティングリソースやストレージ上で複数の顧客に対応した環境を構築し，運用・管理するための技術である．このとき一顧客に対応する一環境をテナントと呼ぶ．マルチテナント技術では，同一の資源を複数のテナントで共有しながらも，テナント同士が互いに影響を与えないように実行プロセスや使用するリソースを分離させるなどの対応を考慮している．

SaaS として提供するシステムは“テナントインタフェース”，“アプリケーション”，“システム基盤”の三つの構成要素に分割することができ，それぞれの構成要素で複数の顧客に対応するための考慮が必要となる（図 1）．マルチテナント技術は，これらの構成要素に対して適用する技術であり，その方法は構成要素ごとに異なる．本稿で述べるマルチテナント技術は，このうち，“アプリケーション”要素に対して適用する技術を範囲とし，マルチテナント技術を利用した場合の考慮点と，アプリケーションの実装パターンについて説明する．

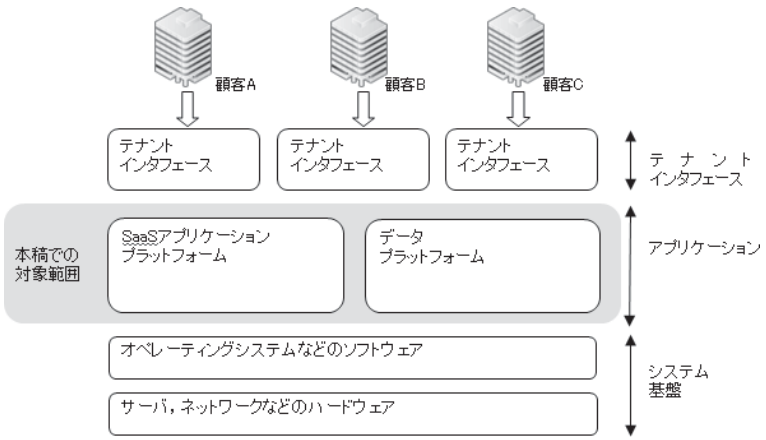

図 1　システム構成と本稿での対象範囲

2.1　マルチテナント技術の特徴と考慮点

マルチテナント技術には，複数の顧客に対して効率的にサービスを提供するために必要な次の特徴がある．

- 使用するコンピューティングリソースやストレージを共有することにより，運用コストを削減する．
- 複数のテナントに提供するアプリケーションを統一することにより，開発負荷，保守・展開時の作業負荷，新規テナント契約時の作業負荷を軽減する．

しかし，これらは，全顧客に対して統一したサービスを提供することによって効率化を図っているため，顧客ごとの要求に対するシステムの柔軟性を犠牲にしている．マルチテナント技

術では，顧客ごとの要求に対応するために，従来の顧客ごとに構築したシステムでのサービス提供に比べて，より複雑な仕組みを検討しなければならない．

マルチテナント技術を利用するに当たって，表1に挙げた項目を考慮し，その適用方針を策定する必要がある．このうち，本稿での対象構成要素である“アプリケーション”にて考慮すべき項目としては，サービス品質保証，データ領域，セキュリティ，テナントカスタマイズの四つが対象となる．本章ではアーキテクチャの観点から特にサービス品質保証とデータ領域についてパターンとして整理する．

表1　マルチテナントで考慮する項目

項目	説明
サービス品質保証	プロセススペースで実行されるプロセスをテナント単位または機能単位に作成し，可視的なメモリーや処理状態，パフォーマンスの変動や動作異常時の影響を分離する必要がある．
データ領域	テナント単位のデータ保持領域を確保してデータを分離する，または，全テナントで一つのデータ保持領域を用意してアプリケーションレベルの実装によりデータをテナント単位に分離する必要がある．
セキュリティ	テナント単位のシステムリソースやデータ間の排他性を確保する必要がある．
テナントカスタマイズ	ユーザーインタフェース，データオブジェクト，業務機能，サービスレベルをテナント単位で調整できる必要がある．
パーソナライズ	ユーザーインタフェース，データオブジェクトをテナント内ユーザーアカウント単位で調整できる必要がある．
データのバックアップとリストア	データベースのバックアップ/リストア，DR，ロールバック，ロールフォワード，診断，インポート/エクスポートなどの機能にテナントごとに対応できる必要がある．
運用状況監視	サービス品質保証と合わせて検討し，プロセス単位で運用状況のモニタリング機能が利用できる，管理パラメータやポリシーの反映ができる必要がある．
テナント管理	テナントインスタンス作成・削除の最適化（低コスト化，短期間）が必要である．
アカウント管理	テナント内ユーザーアカウント作成・削除の最適化（低コスト化，短期間）が必要である．
アプリケーション管理	アプリケーションプラットフォーム上への新規アプリケーションの展開・削除プロセスの最適化（低コスト化，短期間）やアプリケーションのバージョン更新，テナントごとの個別バージョン利用への対応が必要である．

2.2　マルチテナント型アプリケーションの実装パターン

サービス品質保証の視点からアプリケーション実行環境の共有と分離，データ領域の視点からデータベースの共有と分離を検討し，“アプリケーション”要素へのマルチテナント技術適用方針を策定する（図2）．

マルチテナント技術には，選択するそれぞれの技術で，特徴とそれに応じた留意事項がある．共有と分離の選択は，二者択一ではなく，これらの特徴と留意事項を正しく把握し，システム特性，及び想定する利用者規模から共有と分離について判断することが必要である．

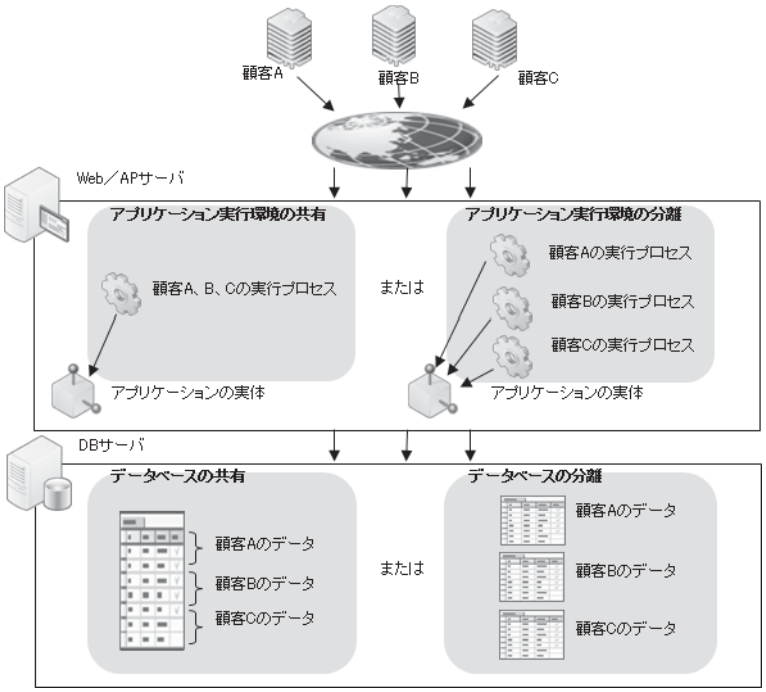

図 2　マルチテナント技術適用方針

2. 2. 1　アプリケーション実行環境の共有と分離

アプリケーションの実体であるバイナリは複数のテナントで共有しつつ，アプリケーションの実行プロセスをすべてのテナントで一つ（共有）にするか，テナントごとに別々（分離）にするかを検討する．それぞれの特徴を表 2 にまとめる．

表 2　アプリケーション実行環境の共有と分離の特徴

アプリケーション実行環境	特徴
共有	・ すべてのテナントでリソースを共有するため，サーバ全体でのリソースの有効活用が可能である． ・ テナント全体でのサービスレベル，サービス提供時間を設定するため，テナント単位でのサービスレベルは低い． ・ テナント全体でのアプリケーション実行環境を設定するため，プロセス実行ユーザーでの権限設定ができない．また，将来のテナントごとの個別バージョン対応ができない． ・ 実行環境の運用管理が容易である．
分離	・ テナントごとのリソースを使用するため，サーバ全体で余剰なリソースを確保しておく必要がある． ・ テナントごとにサービスレベル，サービス提供時間を設定するため，テナント単位のサービスレベルが高い． ・ テナントごとのアプリケーション実行環境を設定するため，プロセス実行ユーザーでの権限設定ができる．また，将来のテナントごとの個別バージョン対応ができる． ・ 実行環境の運用管理が複雑である．

アプリケーション実行環境を共有した場合は，CPU やメモリーなどのリソース共有により，パフォーマンスが不安定になる可能性がある．処理の複雑性にばらつきがあり，ある処理の実行が他の処理の効率に影響を与えたり，一定期間に多数の処理が集中したりするようなアプリケーションでは，処理量の平均化による対処，または機能単位の分離による対処を行い，特定テナントの利用が他のテナントに影響を与えないようにする必要がある．

アプリケーション実行環境を分離した場合は，初期構築や運用コストがテナントごとに必要となるため，これらの自動化や単純化を検討しなければならない．

2.2.2　データベースの共有と分離

すべてのテナントのデータを同一スキーマ[*1]内に保持（共有）し，論理的にデータを分離するか，テナントごとに作成したスキーマ内にデータを保持（分離）し，物理的にデータを分離するかを検討する．それぞれの特徴を表3にまとめる．

表3　データベースの共有と分離の特徴

データベース	特徴
共有	すべての顧客データを同じ領域で管理するため， ・　システム全体でのデータバックアップやメンテナンスなどの運用が容易である． ・　顧客単位でのデータバックやリストアなどの運用が複雑である． ・　顧客単位でのデータに対するアクセス権の設定が複雑である．
分離	顧客ごとのデータ領域で管理するため， ・　システム全体でのデータバックアップやメンテナンスなどの運用が複雑である ・　顧客単位でのデータバックやリストアなどの運用が容易である． ・　顧客単位でのデータに対するアクセス権の設定が容易である．

データベースを共有した場合は，同一データ領域に対してすべてのテナントがアクセス権を持つため，アクセス方法による他テナントデータへのアクセス制御を検討しなければならない．さらに，データのバックアップもテナントごとに行う運用方針を検討しなければならない．

データベースを分離した場合は，初期構築や運用コストがテナントごとに必要となるため，これらの自動化や単純化を検討しなければならない．

3.　マルチテナント型アプリケーションの実装例

3.1　Java 実装例

日本ユニシスが SaaS で提供している Web 型 e ラーニングシステム「RENANDI SaaS」を例に，Java による SaaS アプリケーションの実装例を示す．図3は RENANDI SaaS のシステム概要を示している．RENANDI SaaS はテナントとして顧客ごとにアプリケーションの実行環境とデータベースを分離し，顧客がそれぞれ専用のアプリケーションを使っているように見せている．テナントにはそれぞれに割り当てられた専用の URL でアクセスする．アプリケーションサーバに Apache Tomcat，データベースサーバに PostgreSQL，OS に Linux を利用している．

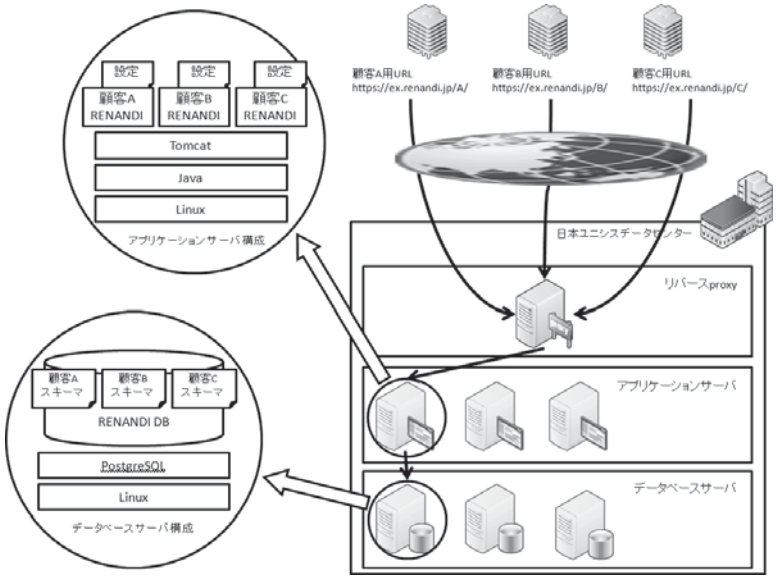

図 3　RENANDI SaaS システム構成図

3. 1. 1　アプリケーション実行環境の分離

テナントは，顧客ごとに異なるデータベースサーバへの接続情報やセッション情報を管理しているため，テナントごとに実行情報の分離が必要である．実行情報の分離には Apache Tomcat の Context を利用している．

具体的には，［テナント名］.xml ファイル（以後，コンテキストファイルとする）を顧客ごとに用意し，［CATALINA_HOME］/conf/Catalina/localhost/（CATALINA_HOME は Apache Tomcat のホームディレクトリ）へ配置する．コンテキストファイルには，このコンテナへアクセスするための URL 情報となる path，プログラムファイルが格納されているパスを指定する docBase，プログラム実行時に利用するテナントごとの情報として Parameter などを記述している．

```
<Context path="/test2" reloadable="false" docBase="/var/renandi/module">
        <Parameter name="tenantName" value="test2" override="false"/>
</Context>
```

すべてのコンテキストファイルの docBase に同じプログラムファイルへのパスを指定することで，プログラムファイルを共通化できる．それにより，テナントごとのバージョン管理に間違いがなくなり，バージョンアップの作業量も軽減できる．

テストや顧客の要望により，通常のサービスとは異なるバージョンのプログラムでサービスを提供する必要がある場合，docBase で異なるプログラムを指定する（図 4）．

一つのサーバで複数のテナントを管理できるが，テナントの数が増えた場合は複数のサーバに分散させる必要がある．RENANDI SaaS では複数のサーバであっても一つのホスト名でアクセスできるようにするため，リバース Proxy を設け，テナントが含まれているサーバへリクエストを送る仕組みとなっている．

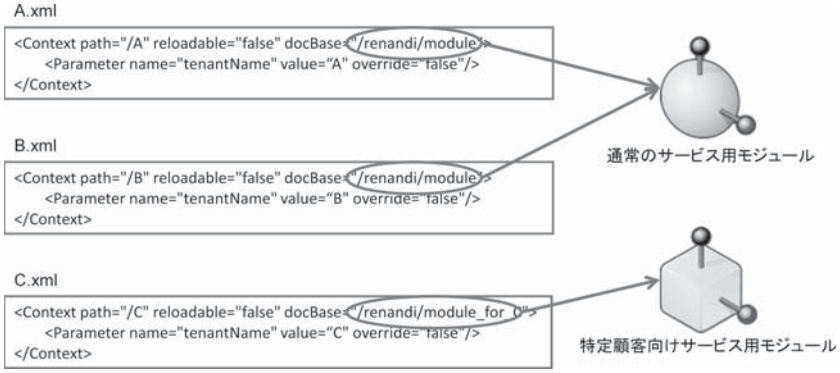

図 4　コンテキストファイル内でのモジュールの指定方法

3. 1. 2　データベースの分離

一つのデータベースサーバに複数の顧客データを保存できるように，RENANDI SaaS ではデータベースに顧客ごとのスキーマを設定している．特定の顧客の実行環境からは，その顧客のデータのみにアクセスが可能となるように実行環境内の接続情報にて制御している．

一部のデータはデータベースではなく，ファイルシステムに格納している．ファイルシステムに格納しているデータは顧客ごとにディレクトリを分けてデータを配置している．アプリケーション自身がどの顧客向けのテナントなのかを認識して正しいデータにアクセスするようになっている．

3. 1. 3　考察

RENADNI SaaS では，基盤として利用している Apache Tomcat の機能を利用することで，マルチテナントではない Web アプリケーションとほぼ同じアーキテクチャでマルチテナントを実現している．この方式ではアプリケーションの構築が容易で，プログラムの共通化や顧客からのカスタマイズ要求にも対応できるなどの利点があるが，注意点もある．docBase で共通のプログラムを指定していても，テナントごとにクラスの情報が読み込まれるため，通常のアプリケーションの運用より多くのメモリーが必要となる．運用してテナントを複数配置してみないと，最適なメモリー値を設定することが難しい．RENANDI SaaS では運用時にヒープサイズを監視し，状況に合わせて設定を変更している．この変更は一時的なサービス停止など顧客に影響を与える．今後のサービスでは顧客に与える影響をなくすための運用や構成，また，複数のアプリケーションを展開した場合にもメモリーの利用量を減らすための方法を検討したい．

3. 2　.NET Framework 実装例

日本ユニシスが提供している特定業種向けの業務サービスを例に，.NET Framework によるマルチテナント型アプリケーションの実装例を示す．提供するサービスは業務の基本機能と個別機能に分類され，顧客は基本機能と利用する個別機能を選択して契約する．サービスの提供形態は RENANDI SaaS と同様である．

図 5 は業務サービスのシステム構成の概要図を示している．Web サーバに IIS（Internet Information Services），データベースに SQL Server，OS に Windows を利用している．

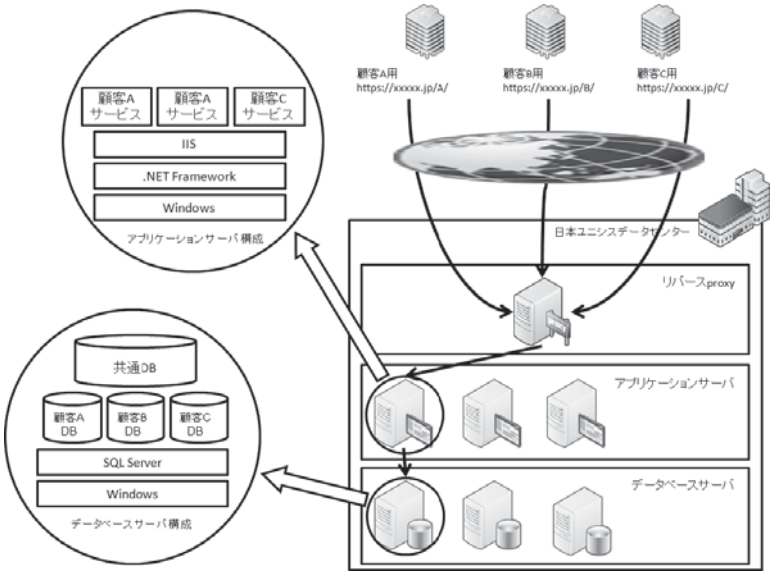

図 5　特定業種向け業務サービスのシステム構成図

業務サービスでは，サービス品質保証が求められ，セキュリティ要件も高かったため，アプリケーションの実行環境を分離，データベースを分離する方式を採用した．

3. 2. 1　アプリケーション実行環境の分離の実現方針

アプリケーション実行環境の分離は IIS のワーカー・プロセス分離モード[*2] を利用することで実現している．その手順は，まず一つの実フォルダに配置したバイナリに対して Web サイト内に顧客数の仮想ディレクトリ[*3] を作成し，それぞれの仮想ディレクトリでアプリケーション（テナント）を構築する．次に，顧客数のアプリケーションプール[*4] を構成し，作成したアプリケーションにアプリケーションプールを割り当てる（図 6）．これにより，一つのフォルダ内のバイナリをすべてのテナントのサービス実行プロセスで共有する運用形態となる．また，アプリケーションプール単位で実行プロセスが作成されるため，仮想メモリーの使用量やプロセスリサイクリングの影響という観点で，テナントごとに独立した安定性を確保することがで

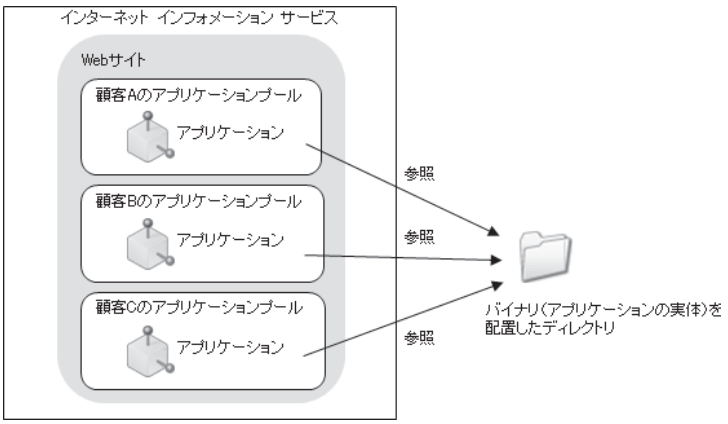

図 6　テナントのサービス実行プロセスとバイナリの参照関係

きる．さらに，アプリケーションプールの動作アカウントとしてテナントユーザーを割り当てることで，アプリケーションはテナントユーザーに割り当てられた権限でのみ動作するようになり，リソースへのアクセス権の設定などを利用してセキュリティを高めることができる．

サービスの個別バージョン対応は，仮想ディレクトリの設定変更により実現している．上記の手順ではバイナリを共有するために，すべての仮想ディレクトリが同一の実ディレクトリを参照するように設定しているが，個別バージョンに対応する場合は，新しいバージョンのバイナリを別のディレクトリに配置し，仮想ディレクトリの参照パスを変更すればよい．

さらに，この業務サービスでは，顧客個別要求に応じたカスタマイズが可能であることが重要な要件であり，カスタマイズ後の保守性等を考慮し，単一の実行コード上に特定の顧客向けのカスタマイズ機能を組み込む実装方式を採用した．この実装方式は，アプリケーション実行環境のメモリー内に保持するテナント情報を利用し，カスタマイズに対応している．その手順は図7のとおりである．

①顧客は，専用のURLにアクセスする．
②アプリケーションは，URLから現在実行中のアプリケーションがどのテナント上で動作しているかを判断し，その情報をメモリーに保持する．
③アプリケーションは，メモリー内のテナント情報を基に，画面に表示する内容や利用できるメニューなど，カスタマイズ情報をデータベースから取得する．
④アプリケーションはカスタマイズ情報を基に，顧客にカスタマイズした画面を表示する．

アプリケーションでは，予めどの範囲をカスタマイズ対象にするかを検討し，カスタマイズに必要なデータのデータベース保持と，そのデータを基にロジックを変更する処理を実装することで，顧客ごとのカスタマイズが可能である．

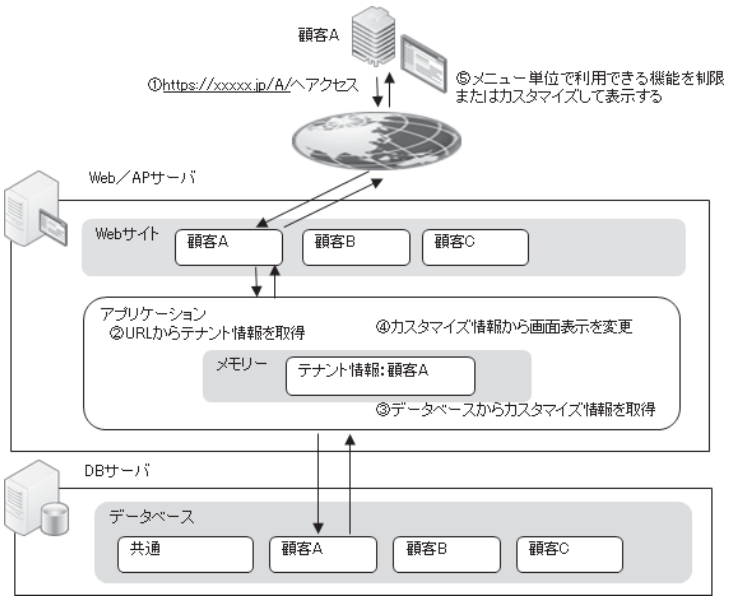

図7　サービス内容のカスタマイズの仕組み

3. 2. 2　データベースの分離の実現方針

業務サービスでは，サービスの契約期間，契約業務機能などシステム管理者が操作する内容を管理する共通データベース（データベース名は共通）を一つ，顧客が操作する業務に関連するマスタデータ，トランザクションデータを管理する顧客データベース（データベース名は顧客 A，顧客 B 等）を顧客数だけ用意し，これらのデータを物理的に分離している．

さらに，データベースのセキュリティを考慮し，それぞれのデータベースに対して顧客に対応したアクセス権を設定し，アプリケーションからデータベースへのアクセス時には SQL Server 認証を行っている．データベースアクセスに使用する接続文字列は顧客単位に管理し，アプリケーションが顧客データベースを正しく識別する仕組みを用意した（図 8）．顧客ごとのデータベースへアクセスする手順は次のとおりである．

まず，顧客ごとに暗号化したデータベース接続文字列をレジストリに登録しておく．データベースアクセス時に，アプリケーションはメモリー内に保持するテナント情報を取得し（図 8 ①），その情報を基にレジストリからデータベース接続文字列を取得する（図 8 ②）．そして，そのデータベース接続文字列を復元し，データベースにアクセスする（図 8 ③）．

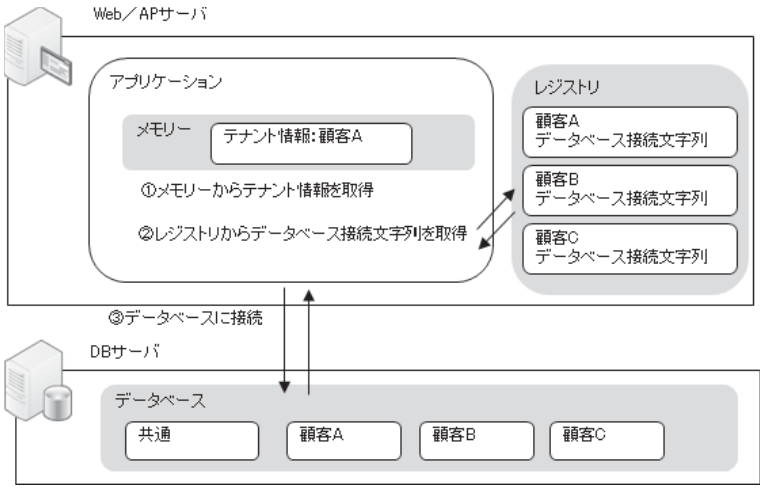

図 8　データベースアクセスの仕組み

3. 2. 3　考察

この業務サービスでは，顧客ごとのアプリケーションの実体であるバイナリを共有し，アプリケーション実行環境を顧客に合わせて分離することで，サービス品質保証を高めつつ，バイナリ管理を一元化できるように考慮した．しかし，サービス提供開始後は，業務の基本機能の追加・修正などすべての顧客に影響する変更と，カスタマイズ機能の追加など特定顧客に影響する変更があり，バイナリ管理の一元化は，前者の場合の作業負荷を軽減することができる．後者の場合は，カスタマイズが発生する最小単位にバイナリを分割しておくことで，変更による他の顧客への影響範囲の極小化，変更の作業負荷軽減ができる．ただし，カスタマイズ要求が高いサービスでは，バイナリの数が多くなる傾向があり，顧客と利用するバイナリの関係が複雑化するため，管理方法を合わせて検討する必要がある．

4. お わ り に

本稿では，SaaSとして提供するためのアプリケーション実装について，マルチテナント対応の考慮点と，それに対する実現方法の例を示した．本稿で示した例は，理論上の仕組みではなく，実際のマルチテナント型アプリケーション構築に適用した仕組みであり，その信用性は高く，マルチテナント型アプリケーションアーキテクチャの基礎となりえるものである．

一方，本稿で述べた内容は，アプリケーション構築に関する事項であり，サービス提供開始後の運用状況監視については触れていない．SaaSとして提供するアプリケーションでは，顧客個々に対するサービス品質保証は重要であり，且つ，そのサービス品質保証のための仕組みは通常のアプリケーションと異なる．今後の課題として，マルチテナント型アプリケーションの運用状況監視の仕組みを検討し，高いサービス品質保証を提供することが挙げられる．

＊1　データベースオブジェクトのまとまりの単位．スキーマはデータを論理的に分割するが，スキーマ単位での管理が可能である．
＊2　ユーザーコードとIISのコアプロセスを完全に分離する技術であり，Webアプリケーションプールごとに個別のワーカー・プロセス（W3WP.EXE）を起動する実行モード．
＊3　実際にはホームディレクトリに含まれていないが，クライアントのブラウザからは含まれているように見えるディレクトリ．
＊4　同一のワーカー・プロセスで実行されるアプリケーションのグループ．

執筆者紹介　中　村　誠　吾（Seigo Nakamura）

2003年日本ユニシス(株)入社．.NET開発においてアプリケーションアーキテクト業務，開発支援に従事．現在は共通利用技術部.NETセンターに所属し，.NET開発の社内開発標準「MIDMOST for .NET Maris」の開発を担当．

田　中　　要（Kaname Tanaka）

1999年日本ユニシス(株)入社．Javaを利用したWeb Application Serverのサポート業務，顧客システムの開発業務に従事．現在はICTサービス基盤開発部に所属し，SaaS型eラーニングシステム「RENANDI SaaS」「LearningCast」の開発，運用を担当．