スレッドローカル変数を用いた Webアプリケーションのマルチテナント化方式の開発

乾敦行　　高野英樹　　秋藤俊介

株式会社 日立製作所 システム開発研究所

1. はじめに

導入コストが不要ですぐに利用できるSaaS(Software as as Service)が注目を集めている．SaaS を開始しようとするアプリケーションパッケージを持つベンダは，コストを削減するために既存のアプリケーションを活用したいニーズがある．通常のアプリケーションは 1 つのユーザ組織で利用することが前提であるため，ユーザ組織ごとにデータを管理していない．SaaS として提供するためには Web アプリケーションおよびデータベースをマルチテナント化する必要がある．マルチテナント化とは，1 つのシステムを複数のユーザ組織で利用できるようにするための技術である．我々は既存の Web アプリケーションをマルチテナント化する方式を開発した．

2. 提案方式

本方式を適用可能な Web アプリケーションの構成は，図 1 に示すように Servlet のアプリケーションサーバが存在し，JDBC でデータベースに接続するものである．図中の X や Y はユーザ組織である．

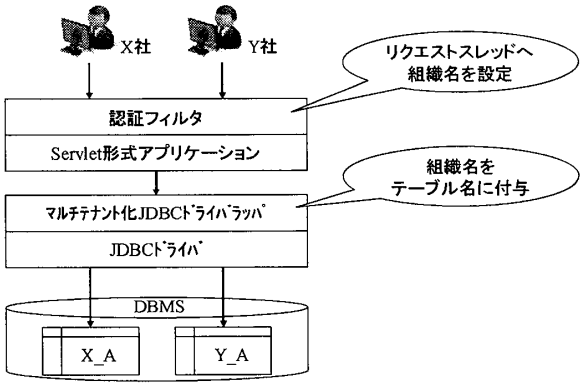

図1　マルチテナント化方式概要

テーブルをユーザ組織ごとに分ける方式を採用した．元のアプリケーションのテーブル A に対応する X_A，X_B テーブルは別途作成する必要がある．開発したのは認証フィルタとマルチテナント化 JDBC ドライバラッパである．本方式では，JDBC ドライバラッパによって X 社からのアクセスであった場合アプリケーションはテーブル X_A を参照し，Y 社からのアクセスであった場合アプリケーションはテーブル Y_A を参照するようになる．以降詳細を説明する．

2.1 認証フィルタ

認証フィルタはユーザ認証をしてリクエスト元のユーザの組織を特定し，その情報を JDBC ドライバラッパに渡すためにリクエストスレッドへ組織名を設定する．リクエストスレッドとは，アプリケーションが 1 つのリクエストを受け付けると割り当てる 1 つのメモリ領域である．このリクエストスレッドは(i)スタック領域，(ii)CPU レジスタ，(iii)スレッドローカル領域からなる．スタック領域は同一プログラム内で使用する局所的な変数を格納する．CPU レジスタはスレッドが現在実行中のプログラムの場所を格納する．スレッドローカル領域は，同一スレッド内で動くプログラム間で共有できる領域である．本方式では，異なるプログラムである認証フィルタとマルチテナント JDBC ドライバラッパ間で認証情報を共有するために，スレッドローカル領域を利用する．スレッドローカル領域への設定は Java 標準 API である ThreadLocal を用いて実装した．

ユーザ認証情報はログイン ID とパスワードであり，ログイン ID には組織名を付加する．例えば X 社の test というユーザの場合，ログイン ID は test@X とする．認証フィルタは@以降が組織名であると判断する．

2.2 JDBC ドライバラッパ

JDBC ドライバラッパは，リクエスト元の組織によって参照するテーブルを切り替えるために，SQL 文に含まれるテーブル名に認証フィルタが設定した組織名を付与する．SQL 文を実行するのは java.sql.Statement インタフェースの execute 系メソッドと， java.sql.PreparedStatement インタフェースの execute 系メソッドである．そこで，

Development of multi tenant method for web applications using thread local variables
Atsuyuki Inui, Hideki Takano, Shunsuke Akifuji
Systems Development Laboratory, Hitachi, Ltd.

これらのメソッドをオーバーライドして JDBC ドライバに組織名を付与する処理を追加したラッパークラス sdl.Statement を作成した．Java 標準 API では，Statement オブジェクトは java.sql.Connection オブジェクトから取得し，java.sql.Connection オブジェクトは java.sql.Driver クラスから取得することになっている．そこで sdl.Statement オブジェクトを取得できるように，java.sql.Connection，java.sql.Driver をそれぞれラップしたクラスを開発した．

組織名をテーブル名に付加するアルゴリズムは以下の通りである．

1. SQL 文を空白文字で分解
2. info, from, table の次の単語に組織名を付加 (SQL の構文規則上これらの後に続く単語はテーブル名であるため)
3. ドットの直前の単語に組織名を付加(SQL の構文規則上ドットの直前の単語はテーブル名であるため)
4. 単語を空白文字で結合

例えば「select * from users」という SQL 文に対してこのアルゴリズムを適用すると「select * from X_users」となる．

開発したマルチテナント化技術を利用するためには，認証フィルタと JDBC ドライバラッパを使用する設定をアプリケーションサーバに対して行えば良い．

3. 評価

オープンソースの CRM である Eberom と JEE のサンプルアプリケーションである Pet Store を本方式でマルチテナント化し，工数を評価した．Eberom については性能も評価した．

3.1 工数評価

それぞれの工数は表 1の通りである．5,000 行を超える大規模なアプリケーションに対しても，数行の変更でマルチテナント化することができた．Pet Store よりも Eberom の変更量が多いのは，元々あった認証処理を削除する分のためである．

表1　工数比較

	変更量	オリジナルのコード量
Eberom	20 行	14,600 行
Pet Store	3 行	5,400 行

3.2 性能評価

マルチテナント化した Eberom の性能を測定した．Eberom のデータベースは 1 組織あたり 36 のテーブルを持つ．同時接続するユーザ数は 55, 110, 165, 220 の 4 通りで，1 ユーザあたり Read:Write=10:1 のリクエストを 50 回行い，スループットを計測した．結果は図2 の通りである．

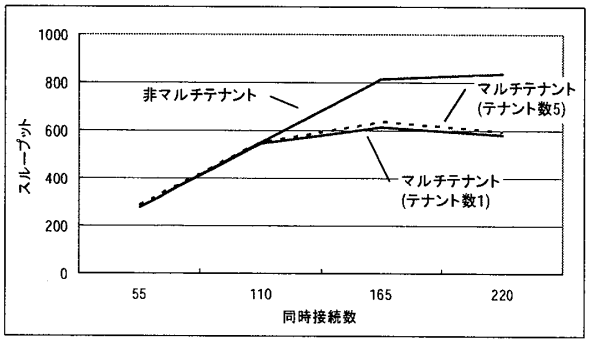

図 2 スループット比較

図 2 より，同時接続数が 110 まではマルチテナント化によるスループットの低下は発生しないことが分かった．またテナント数によってスループットは変わらないことを確認した．

次にレスポンスタイムの比較結果を図 3 に示す．

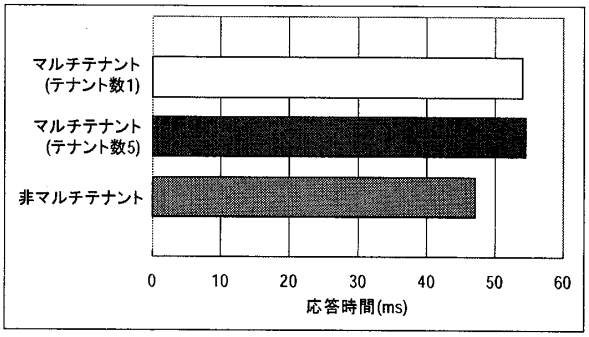

図 3 レスポンスタイム比較

Read:Write=4:1 の割合でリクエストを送信した．マルチテナント化後のレスポンスタイムは約 15%遅くなった．これは認証フィルタにおける処理と JDBC ドライバラッパの処理を追加したためであると考えられる．

4. おわりに

Web アプリケーションのマルチテナント化方式について述べた．本方式は組織名を JDBC まで渡す方法として ThreadLocal を用いている．そのため，1 つのリクエストを処理するのは 1 つのスレッドであるアプリケーションに対してのみ有効である．多くのアプリケーションでは 1 つのスレッドで動作するが，適用可能なアプリケーションを増やすためには ThreadLocal の制約をなくすことが課題となる．