修士論文

Web-DB インタラクション測定

辰己　圭介

平成 13 年度入学

島根大学大学院　総合理工学研究科修士課程

数理・情報システム学専攻　計算機科学講座

指導教官　田中章司郎

目次

第１章　はじめに

研究の背景

近年、インターネットが急速に普及している[1]。2002 年 11 月末現在のブロードバンドサービス(DSL、FTTH、CATV)の加入者は累計で 719 万人、またナローバンドサービス(電話回線等を利用したダイアルアップ型接続)の加入者は累計で 2129 万人にも上っている。これは 3 年前と比べると、約 2.5 倍にまで増加している。このことは、多くの家庭でインターネットの普及が進んでいるといえる。

それに伴って、さまざまな企業がプロモーション活動の一環として Web サイトの開設をしてきている。また Amazon や楽天など、インターネット販売を行う会社も増えてきている。そこでは、データの整合性や複数ユーザの同時アクセスができるといった利点で使用されている、DataBase を利用した Web ページが多く用いられている。しかし、それぞれのサイトによって、Web ページのレスポンスの差に違いが生じているのが現状である。

本研究の概要

本研究では、レスポンスの違いがどのような Web ページで生じるのか、その原因を追究するため、クライアントにモニタを設置し、DataBase を用いた Web ページの表示に要する時間を計測した。

[1] 総務省調べ（参考文献・URL[1]）

第 2 章　Web ページ

本研究では、さまざまな Web ページの計測を行った。作成した Web ページは大きく DataBase と接続するもの・しないものと、2 種類に分けることができる。DataBase に接続しないものは、HTML・Java Servlet を用いて、文字と画像との組み合わせで製作した。DataBase に接続するものは JDBC(Java DataBase Connectivity)を用いて Java Servlet で製作した。本章では、DataBase に接続する Web ページの製作について述べる。

2.1　JDBC ドライバ [2]

2.1.1　JDBC ドライバとは

JDBC は、Java で書かれたアプリケーションやアプレットがリレーショナルデータベースにアクセスできるように用意された Java のクラスの集まりである。

JDBC には 2 つの API(JDBC API、JDBC ドライバ API)がある。JDBC API は DataBase 製品の種類にかかわらず基本的には同一なので、アプリケーション作成者は DataBase 製品の違いにあまり煩わされることなく、アプリケーションロジックに専念することができる。また、JDBC ドライバ API は DataBase に直接アクセスする。DataBase による違いは JDBC ドライバが吸収する。よって JDBC ドライバ API は DataBase ごとに用意しなければならない。

JDBC ドライバには 4 種類のタイプ(JDBC-ODBC ブリッジ、ネイティブ API、JDBC ネット、Java ネイティブプロトコル)がある。JDBC-ODBC ブリッジは、内部で ODBC ドライバを使って高速にアクセスし、プラットフォームに依存した ODBC ドライバが必要である。ネイティブ API は、内部でネイティブ API のメソッドを使って高速にアクセスし、プラットフォームに依存した専用ドライバが必要である。JDBC ネットは、すべて Java で汎用ネットワークプロトコルを使ってアクセスし、プラットフォーム独立だがネットワークプロトコルのコネクタが必要である。Java ネイティブプロトコルは、すべて Java でネイティブプロトコルを使ってアクセスプラットフォーム独立なため、サーブレットやアプレットなどで利用する。

本研究で用いた PostgreSQL に付属する JDBC ドライバは、Java ネイティブプロトコル(ダイレクトドライバ)と呼ばれるもので、Java だけで記述されており、しかも ODBC などに頼らず直接 DataBase に接続できるタイプである。このタイプの JDBC の利点は、

[2] 参考文献・URL [2]より引用

native メソッドを含まないのでアプレットからも利用できること、プラットフォームを選ばないため可搬性が高いこと、ODBC などを経由しないので性能がいいことなどが上げられる。欠点としては、すべて Java で記述しなければならないので、クラスライブラリが大きくなりがちであることである。

2.1.2　Java 環境・JDBC ドライバの導入

本研究で利用した環境は以下のようになる。

OS：Windows 2000 Server

Java：Java 2 SDK、Standard Edition Version 1.4.0_01

→ http://java.sun.com/よりダウンロード

JDBC ドライバ：JDBC3

→ http://jdbc.postgresql.org/よりダウンロード

環境変数の設定

[スタート]から[設定]→[コントロールパネル]→[システム]の順に選択する。[システム環境変数]の[PATH]に java をインストールした場所を追加する。インストールする際にディレクトリを変えていなければ、C:¥j2sdk1.4.0_01¥bin のようになる。パスには、複数のディレクトリをセミコロン(;)で区切って並べて指定する。その後、[適用]をクリックする。

JDBC ドライバも、上記と同様に[CLASSPATH]に追加する。追加するのは postgre.jar の置いてあるアドレスをファイル名まで指定する。また[CLASSPATH]には、カレントディレクトリを含めるという意味で“.”も追加する。

2.1.3　DataBase との接続

JDBC API を使うためには、JDBC のクラスを import しておかなければならない。これは import java.sql.*;とファイルの先頭に書いておけばよい。しかしこのままでは JDBC は使えない。なぜならば実際に DataBase にアクセスする JDBC ドライバがまだ存在していないからである。次に PostgreSQL JDBC ドライバをロードする。本研究では PostgreSQL JDBC ドライバ以外は使わないため、以下の Class.forName を使用した。

```
1:try {
2:    Class.forName(“org.postgresql.Driver”);
3:    // Connect to database
4:} catch (ClassNotFoundException ex) {
5:    // 例外処理を記述
6:}
```

JDBC のドライバがロードできたら、次に DataBase に接続する。これは JDBC API の DriverManager.getConnection メソッドを使う。

Connection conn = DriverManager.getConnection(url, usr, pwd);

引数の usr、pwd は String 型でユーザ名、パスワードを渡す。url も String 型であるが、これは WWW の URL のような形式で接続先の DataBase に関する情報を表現する。これについては前述のように

jdbc:postgresql:databasename

jdbc:postgresql://hostname/databasename

jdbc:postgresql://hostname:port/databasename

の 3 つの形式がある。本研究では jdbc:postgresql://localhost:5432/test のように指定した。

2.1.4　サーブレットとは

サーブレットはサーバで動く Java のプログラムである。クライアントの Web ブラウザから要求があると、サーブレットのプログラムが HTML やその他のリソースを動的に生成して結果を Web ブラウザに返している。

2.1.5　サーブレット環境の導入

本研究で利用した環境は以下のようになる。

サーブレットコンテナ：Tomcat 3.3.1

→http://jakarta.apache.org/tomcat/よりダウンロード

2.2　テーブルの製作

本研究では、パソコンショップの販売システムをモデルとし、5 つのテーブルを定義した。作成者は root としたが、これらのテーブルにアクセスすることのできる DataBase ユーザは制限していない。本研究では、すべてのデータベースのアクセスは、この DataBase ユーザ root を使用している。またテーブルごとの検索処理能力を高めるためのインデックスは用いていない。以下が、作成したテーブルである。

customer・・・顧客に関するテーブル

列名	データ型		項目
C_ID	整数値型		顧客 ID
C_FNAME	文字列型	最大長 20	性
C_LNAME	文字列型	最大長 20	名
C_ZIP	整数値型		郵便番号
C_ADDR1	文字列型	最大長 50	住所 1
C_ADDR2	文字列型	最大長 50	住所 2
C_PHONE	整数値型		電話番号
C_EMAIL	文字列型	最大長 50	メールアドレス

item・・・商品に関するテーブル

列名	データ型		項目
I_ID	整数値型		商品 ID
I_TITLE	文字列型	最大長 60	商品名前
I_M_ID	整数値型		商品のメーカーID
I_SUBJECT	文字列型	最大長 10	種類
I_PRICE	整数値型		価格
I_IMAGE	文字列型	最大長 40	画像 URL

maker・・・メーカーに関するテーブル

列名	データ型		項目
M_ID	整数値型		メーカーID
M_NAME	文字列型	最大長 50	メーカー名
M_URL	文字列型	最大長 50	メーカーURL

orders・・・顧客ごとの注文に関するテーブル

列名	データ型	項目
O_ID	整数値型	注文 ID
O_DATE	日付	注文された日付
O_C_ID	整数値型	注文した顧客 ID

order_item・・・注文に対する顧客・商品に関するテーブル

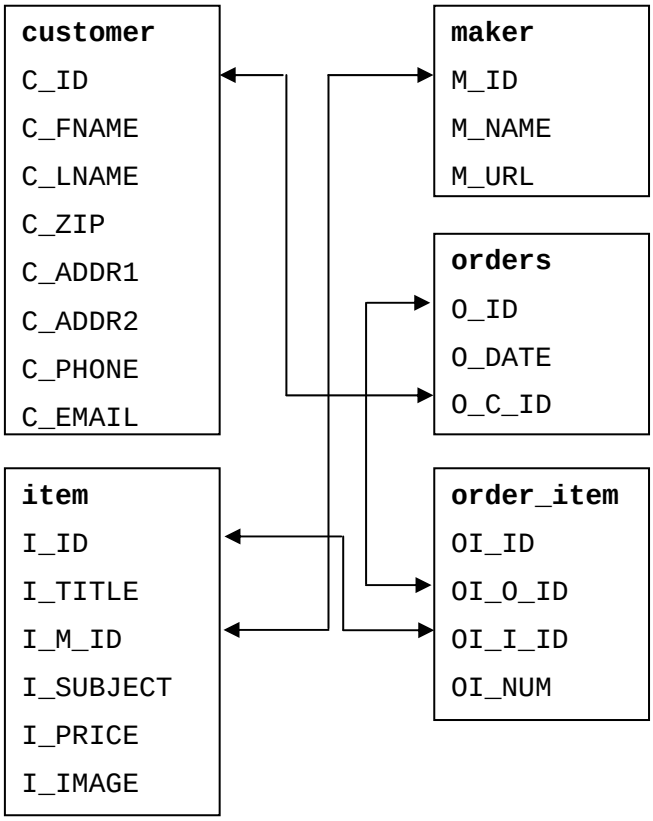
列名	データ型	項目
OI_ID	整数値型	顧客別注文の ID
OI_O_ID	整数値型	注文 ID
OI_I_ID	整数値型	注文した商品 ID
OI_NUM	整数値型	注文した商品の個数

計測の種類は DataBase の検索で分けるため、データの行数は以下のようにした。

データ量	多	中	少
顧客テーブル	5000	500	50
商品テーブル	20000	2000	300
メーカーテーブル	60	60	60
注文テーブル	2000	200	20
顧客別注文テーブル	6050	616	62

顧客別注文テーブルのデータ量に一貫性がないように思えるが、商品を注文した顧客は、プログラム的にランダムで 1 種類から 5 種類の商品を注文するという様にしたためである。
また、各テーブルの相関関係は次のようになる。

図 2.1　テーブルの相関関係



2.3 Web ページデザイン

本研究では以下の表のようにページを製作した。

表 2.1 製作した Web ページのデータ量

DB 接続なし		文字・少	10,000 bytes
		文字・多	100,000 bytes
		画像・小	5,244 bytes
		画像・大	22,747 bytes
		文字・少　画像・小	62,440（10,000 + 52,440）bytes
		文字・少　画像・大	464,890（10,000 + 454,890）bytes
		文字・多　画像・小	152,440（100,000 + 52,440）bytes
		文字・多　画像・大	545,890（100,000 + 454,890）bytes
DB 接続あり	データ量 少	文字・少	4,781 bytes
		文字・多	9,126 bytes
		文字・少　画像・小	27,401（1,187 + 26,214）bytes
		文字・少　画像・大	114,375（1,087 + 113,288）bytes
		文字・多　画像・小	162,080（4,702 + 157,378）bytes
		文字・多　画像・大	684,480（4,672 + 679,808）bytes
	データ量 中	文字・少	4,782 bytes
		文字・多	86,977 bytes
		文字・少　画像・小	27,454（1,121 + 26,333）bytes
		文字・少　画像・大	114,331（1,116 + 113,215）bytes
		文字・多　画像・小	162,561（4,748 + 157,813）bytes
		文字・多　画像・大	685162（4,718 + 680,444）bytes
	データ量 多	文字・少	4,782 bytes
		文字・多	865,477 bytes
		文字・少　画像・小	27272（1,083 + 26,189）bytes
		文字・少　画像・大	114,795（1,078 + 113,717）bytes
		文字・多　画像・小	162,650（4,834 + 157,816）bytes
		文字・多　画像・大	686,128（4,804 + 681,324）bytes

DataBase に接続しない Web ページは、ただ単に文字の羅列により製作している。
DataBase に接続する Web ページについては、次節で述べる。

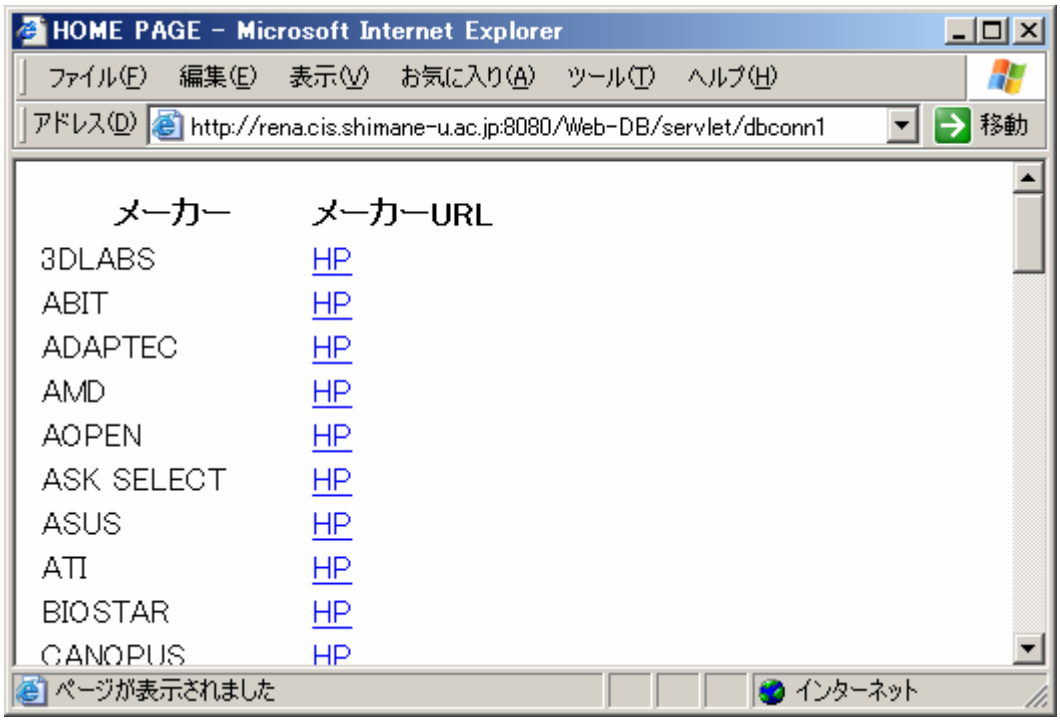
2.4　Web ページ上の操作内容

DataBase と接続する Web ページに使用した SQL コマンドは、テーブルから情報を得る SELECT を使用した。

以下は、製作した Web ページと使用した SQL 文である。

・文字少

図　DataBase に接続する Web ページ（文字少）

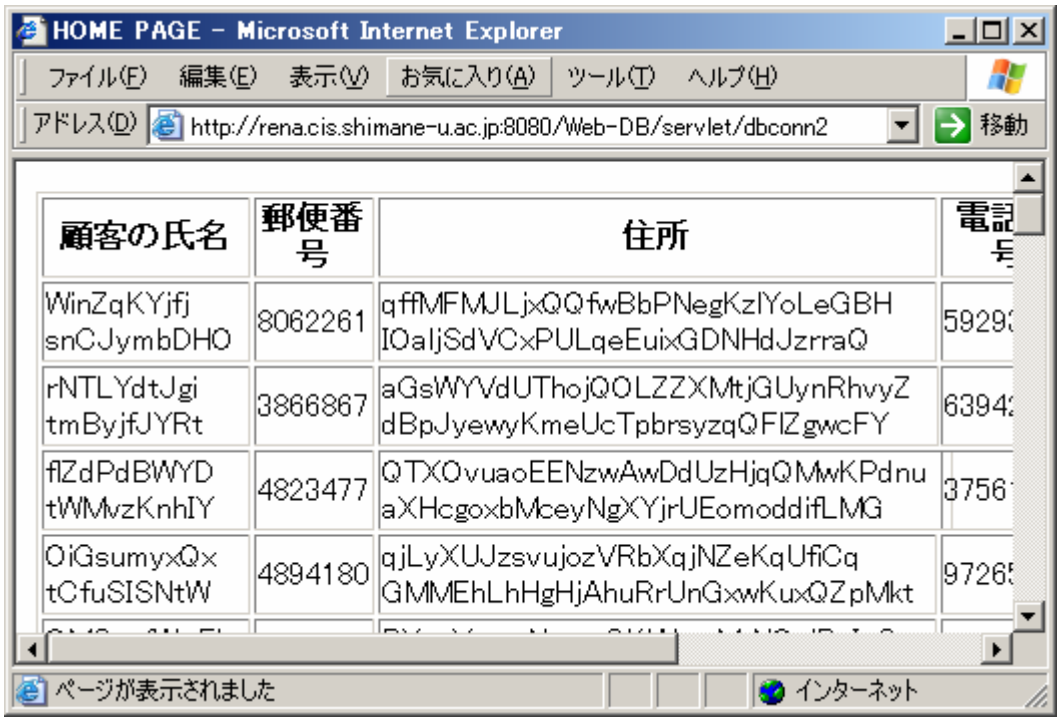


SELECT ＊ FROM maker

メーカーテーブルすべて表示する。

・文字多

図　DataBase に接続する Web ページ（文字多）

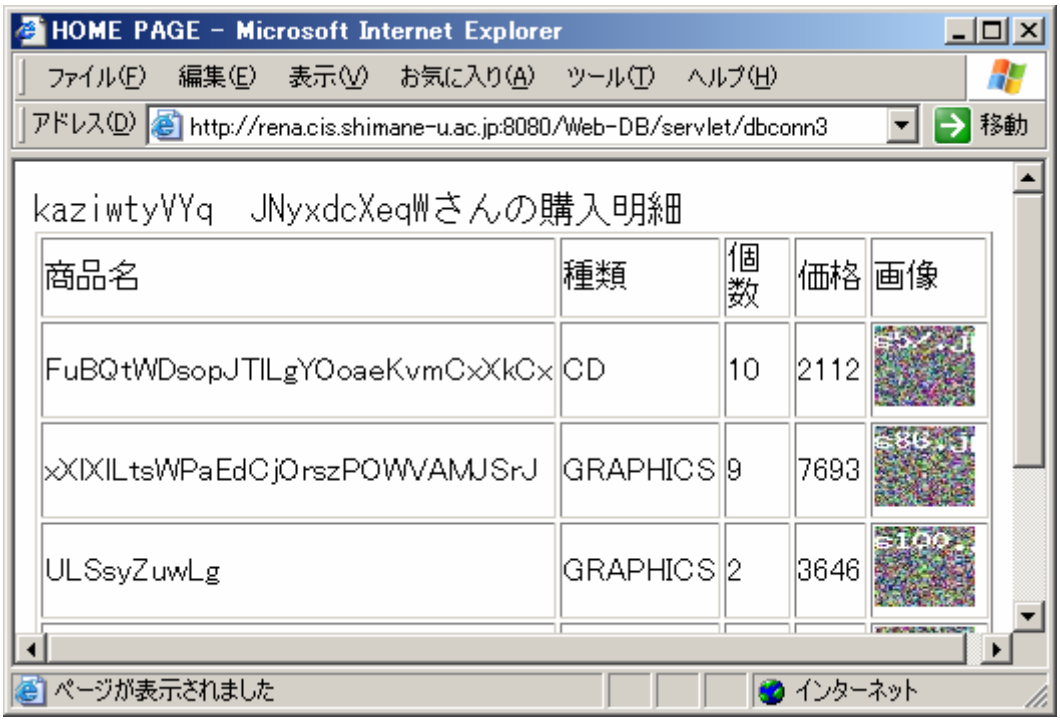


SELECT ＊ FROM customer

顧客テーブルすべて表示する。

・文字小＋画像

図　DataBase に接続する Web ページ（文字少＋画像）



SELECT c_fname, c_lname FROM customer WHERE c_id = ?

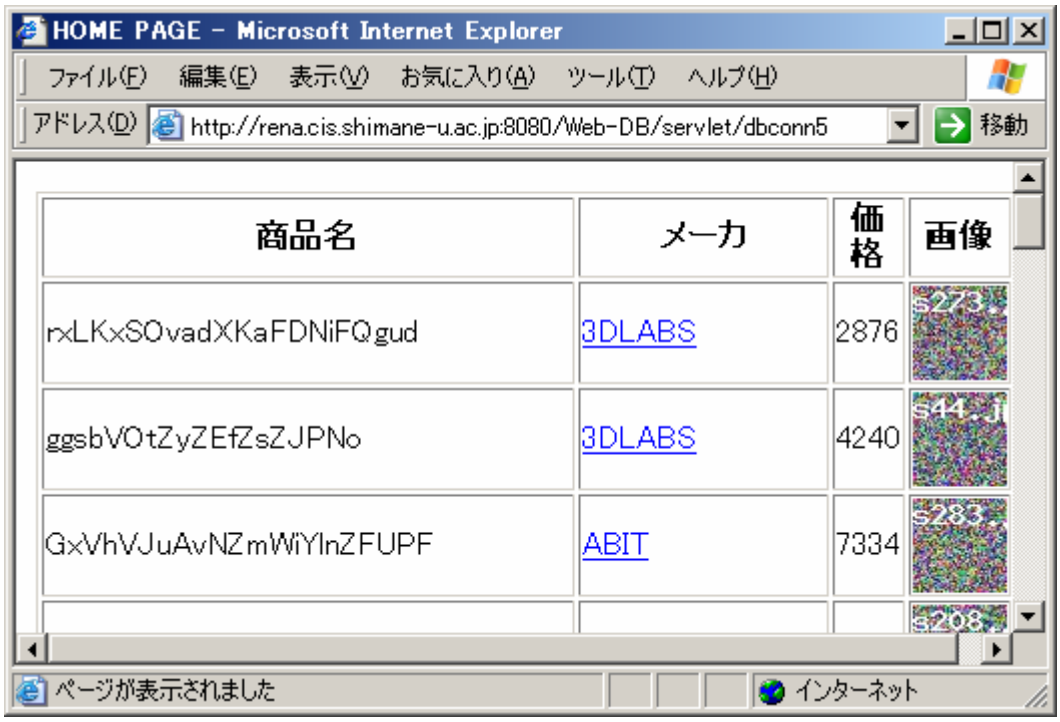
顧客テーブルから顧客の ID が?であるものを検索し、そのときの顧客の氏名を表示する。

SELECT i_title, i_subject, oi_num, i_price, i_image FROM item, orders, order_item
WHERE orders.o_id = order_item.oi_o_id AND order_item.oi_i_id = item.i_id AND
orders.o_c_id = ?

注文テーブルから注文した顧客の ID が?であるものを検索
・注文テーブルの注文 ID と顧客別注文テーブルの注文 ID が同じであるもの、
・顧客別注文テーブルの注文した商品の ID と商品テーブルの商品 ID が同じであるもの
以上の条件を満たす、商品テーブルの商品の名前・商品の種類・商品の価格・商品の画像 URL、顧客別注文テーブルから注文した商品の個数を表示する。

・文字大＋画像

図　DataBase に接続する Web ページ（文字少）



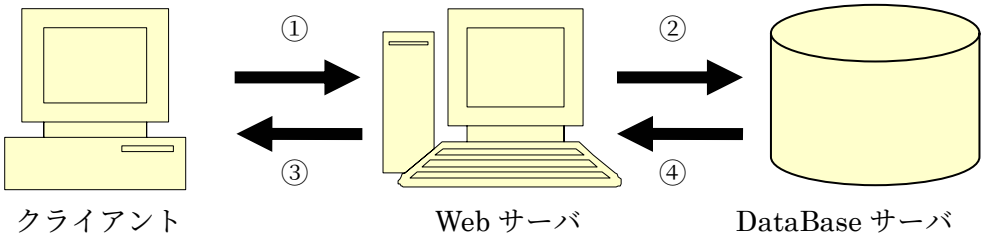
SELECT m_name, m_url, i_title, i_price, i_image FROM item, maker WHERE maker.m_id = item.i_m_id AND i_subject = 'CPU' LIMIT 30

商品テーブルから商品の種類が CPU であるものを検索
・メーカーテーブルからメーカーID と商品テーブルから商品のメーカーID が同じであるもの
以上の条件を満たす、商品テーブルの商品の名前・商品の価格・商品の画像 URL、メーカーテーブルのメーカー名・メーカーURL から 30 行のみを表示する。

第 3 章　モニタの設計

Web ページは、サーバとクライアント間に接続を確立してから、クライアントからのリクエスト（①）に対して、サーバがレスポンスを返す（③）という形で、データのやりとりを行う。また Java Servlet の Web ページは JDBC ドライバのロード・DataBase との接続・SQL 文の実行（②）、処理結果の取得（④）を行う。

図 3.1　システム概要



本研究では、Visual Studio 6.0 Enterprise を用いて作成した。本章では作成したモニタについて述べる。

3.1　CHtmlView クラス [3]

CHtmlView クラスは、MFC(Microsoft Foundation Class)のドキュメント/ビューアーキテクチャにおける Web Browser コントロールの機能を提供している。Web Browser コントロールは、ユーザが World Wide Web 上のサイトだけでなく、ローカルファイルシステムおよびネットワーク上のフォルダを参照できるウィンドウである。Web Browser コントロールはハイパーリンクによるジャンプ、Uniform Resource Locator(URL)ナビゲーションのサポート、および履歴リストを保持する。

MFC アプリケーションでの CHtmlView クラスの使い方
MFC の標準フレームワークアプリケーション(SDI[4]、MDI[5])では、通常、ビューオブジェクトは特別な用途のクラス群の機能を継承している。これらのクラスはすべて CView の派生クラスであり、CView では提供されない特別な用途の機能を提供している。

アプリケーションのビュークラスを CHtmlView から派生させると、Web Browser コント

[3] 参考文献・URL [3]から引用
[4] Single Document Interface　同時に１つのドキュメントのみ処理するアプリケーション
[5] Multiple Document Interface　同時に複数のドキュメントを処理するアプリケーション

ロール付きのビューになる。これにより、アプリケーションを Web ブラウザとして使用できる。Web ブラウザ形式のアプリケーションを作成するには、[MFC アプリケーションウィザード]を使用し、CHtmlView をビュークラスとして指定した方がよい。
Web Browser の ActiveX コントロール(つまり CHtmlView)を使用できるのは、Internet Explorer 4.0 以降がインストールされた Windows NT version 4.0 以降で実行されるプログラムだけである。CHtmlView は、Web や HTML ドキュメントにアクセスするアプリケーション用に設計されている。CHtmlView の以下のメンバ関数は、Internet Explorer アプリケーション専用である。これらの関数は Web Browser コントロールにも継承されるが、何も機能しない。

3.2 モニタの設計

3.2.1　本体の作成

　前節で述べたように、本研究では MFC アプリケーションウィザードを用いた。ファイルから新規作成を選択。プロジェクトから MFC AppWizard(exe)を選び、プロイジェクト名をつける。(本研究では WebBrowse とした)
AppWizard での変更すべき点

ステップ１　作成するアプリケーションの種類	SDI
ステップ４　ツールバーの外見	Internet Explorer ReBar
ステップ６　基本クラス	CHtmlView

この状態で必要最低限なブラウザ機能は完成する

3.2.2　アドレスバーの追加

　3.2.1 にできたブラウザのダイアログバーに URL を打ち込むためのエディットボックスを追加（名前を IDC_EADDRESS とする）する。

・関数の宣言
MainFrm.h に下記を追加する。
　void OnNewAddress()　自作関数

・本文
MainFrm.cpp に下記を追加する。

```
1:void CMainFrame::OnNewAddress()
2:{
3:    CString sAddress;
```

```
4:  m_wndDlgBar.GetDlgItem(IDC_EADDRESS)->GetWindowText(sAddress);
5:  ((CWebBrowseView*)GetActiveView())->Navigate(sAddress);
6:}
```

4 行目　GetWindowText 関数を使ってエディットボックスに入力されたテキストを取得し、そのテキストを sAddress 変数に代入する。

5 行目　GetActiveView 関数からの戻り値のポインタを CWebBrowseView クラスへのポインタにキャストする。このポインタを介してビュークラスの Navigate 関数を呼び、エディットボックスに入力された URL を渡す。

また、MainFrm.cpp に WebBrowseDoc.h、WebBrowseView.h をインクルードする。

3.2.3　計測開始地点の追加

3.2.1 にできたブラウザのダイアログバーに計測を開始したときの時間を表示するためのスタティックテキストを追加（名前を IDC_STATICTIME1 とする）する。

・関数･変数の宣言

MainFrm.h に下記を追加

double start_time　自作変数

void GetAddress()　自作関数

WebBrowseView.h に下記を追加

```
void OnBeforeNavigate2(LPCTSTR lpszUrl, DWORD nFlags, LPCTSTR
  lpszTargetFrameName, CByteArray& bePostedData, LPCTSTR lpszHeaders,
  BOOL* pbCancel)　MFC ライブラリ関数
```

・本文

WebBrowseView.cpp に下記を追加する。

```
1:void CWebBrowseView::OnBeforeNavigate2(LPCTSTR lpszUrl, DWORD nFlags,
      LPCTSTR lpszTargetFrameName, CByteArray& bePostedData, LPCTSTR
      lpszHeaders, BOOL* pbCancel)
2:{
3:  ((CMainFrame*)GetParentFrame())->GetAddress();
4:}
```

3 行目　OnBeforeNavigate2 関数を用いて、GetAddress 関数を呼び出す。

MainFrm.cpp に下記を追加する。

```
1:void CMainFrame::GetAddress()
2:{
3:   start_time = GetTickCount();
4:   CString tmp1;
5:   tmp1.Format( "%.0f", start_time);
6:   m_wndDlgBar.GetDlgItem(IDC_STATICTIME1)->SetWindowText((LPCTSTR)tmp1);
7:}
```

3 行目　GetTickCount 関数を用いて、今までのシステム起動時間を start_time に代入する。

6 行目　SetWindowText 関数を用いて、スタティックテキスト内の文字列を start_time に変更する。

3.2,4　計測終了地点の追加

3.2.1 にできたブラウザのダイアログバーに、計測を終了したときの時間・計測の終了－開始の差の時間を表示するためのスタティックテキストを追加（名前を IDC_STATICTIME2、IDC_STATICTIME3 とする）

・関数・変数の宣言

MainFrm.h に下記を追加

double stop_time　自作変数

void SetAddress()　自作関数

WebBrowseView.h に下記を追加

void OnDocumentComplete(LPCTSTR lpszUrl)　MFC ライブラリ関数

本文

WebBrowseView.cpp に下記を追加する。

```
1:void CWebBrowseView::OnDocumentComplete(LPCTSTR lpszUrl)
2:{
3:   ((CMainFrame*)GetParentFrame())->SetAddress(lpszUrl);
4:}
```

3 行目　OnDocumentComplete 関数を使って Web ページの URL を引数として渡す。

MainFrm.cpp に下記を追加する。

```
1:void CMainFrame::SetAddress(LPCTSTR lpszURL)
2:{
3:   m_wndDlgBar.GetDlgItem(IDC_EADDRESS)->SetWindowText(lpszURL);
4:
5:   stop_time = GetTickCount();
6:   CString tmp2;
7:   tmp2.Format( "%.0f", stop_time);
8:   m_wndDlgBar.GetDlgItem(IDC_STATICTIME2)->SetWindowText((LPCTSTR)tmp2);
9:
10:   double dif_time;
11:   dif_time = stop_time - start_time;
12:   CString tmp3;
13:   tmp3.Format( "%.0f", dif_time);
14:   m_wndDlgBar.GetDlgItem(IDC_STATICTIME3)->SetWindowText((LPCTSTR)tmp3);
15:}
```

5 行目　GetTickCount 関数を用いて、今までのシステム起動時間を stop_time に代入する。

8 行目　SetWindowText 関数を用いて、スタティックテキスト内の文字列を stop_time に変更する。

11 行目　stop_time - start_time の計算をすることにより、リクエストしてから表示されるまでの時間になる。それを dif_time に代入する。

14 行目　SetWindowText 関数を用いて、スタティックテキスト内の文字列を dif_time に変更する。

また、WebBrowseView.cpp に、MainFrm.h をインクルードする。

以下は、今回使用した CHtmlView クラスのメンバ関数である。

CHtmlView::Navigate2

URL で指定されたリソース、または完全パスで指定されたファイルに移動する。

CHtmlView::OnBeforeNavigate2

Web ブラウザでのナビゲーション開始前にイベントを生成するために、フレームワークによって呼び出される。

CHtmlView::OnDocumentComplete

ドキュメントが READYSTATE_COMPLETE 状態に達したことをアプリケーションに通知するために、フレームワークによって呼び出される。

図 3.1 は作成したモニタである。

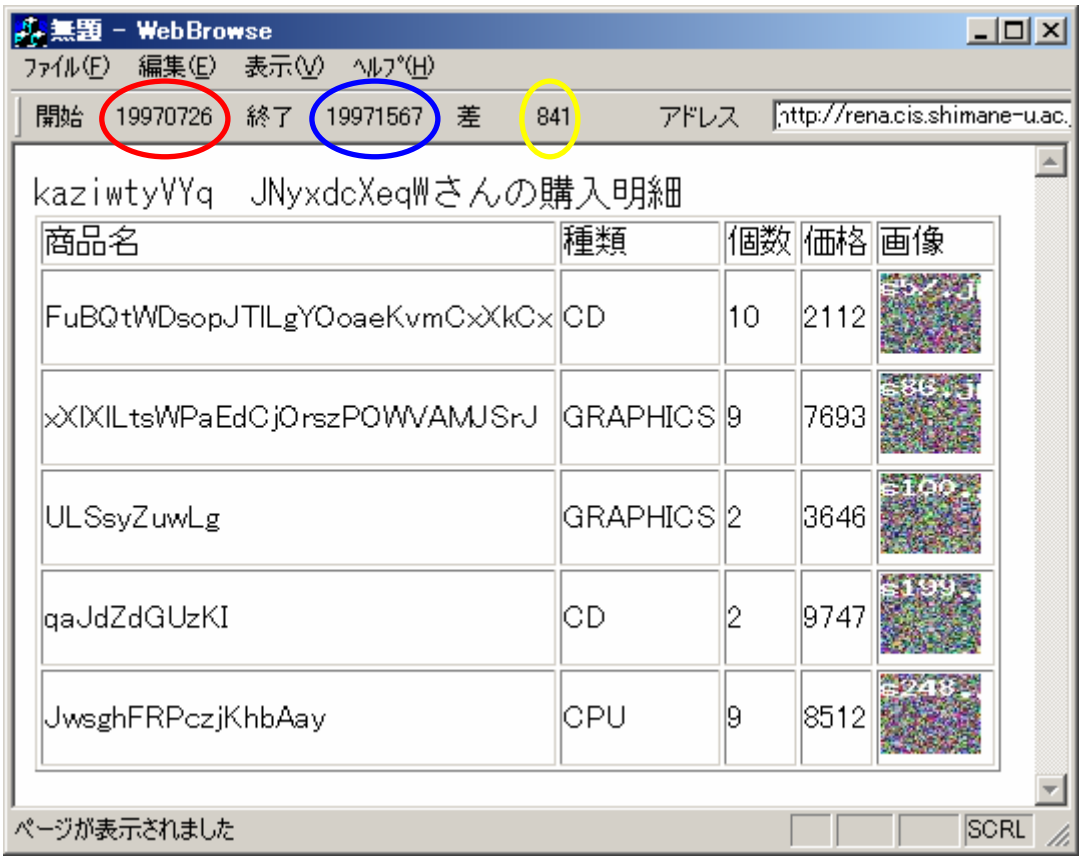


図 3.1　作成したモニタ

図 3.1 の、赤円が計測開始時の時間、青円が計測終了時の時間、黄円がその差である。

3.3　計測方法

アドレスバーに、計測したい URL を打ち込む。Web ページが表示されたときに、作成

した差の表示を見ることにより、リクエストしてからレスポンスが返ってくるまでの時間がわかる。

Web ページは表示させると、キャッシュにその Web ページの情報が残る。同じ Web ページを再び表示させると、キャッシュを読みにいくため、表示するのに要する正しい時間は計測されない。そのため、同じ Web ページを読み込む場合は、C:¥Documents and Settings¥ユーザ名¥Local Settings¥Temporary Internet Files の中にあるファイルを削除してから新たなページを表示させる必要がある。

第 4 章　実験結果

本研究で使用したハードウェア・ソフトウェア（一部前述あり）は以下のようになる。また、図 4.1 にネットワーク関係を示す。

・Web サーバ

ハードウェア

CPU	Pentium3 733MHz
Memory	256M
LAN カード	Intel 8255x-based PCI Ethernet Adapter (10/100)

ソフトウェア

OS	Windows2000 Server
DBMS	PostgreSQL 7.2.1 native
Compiler	Java 2 SDK、Standard Edition Version 1.4.0_01
Servlet Container	Tomcat 3.3.1

・クライアント

ハードウェア

CPU	Celeron 1.3GHz
Memory	512M
LAN カード	Sis 900-Based PCI Fast Ethernet Adapter

ソフトウェア

OS	Windows XP

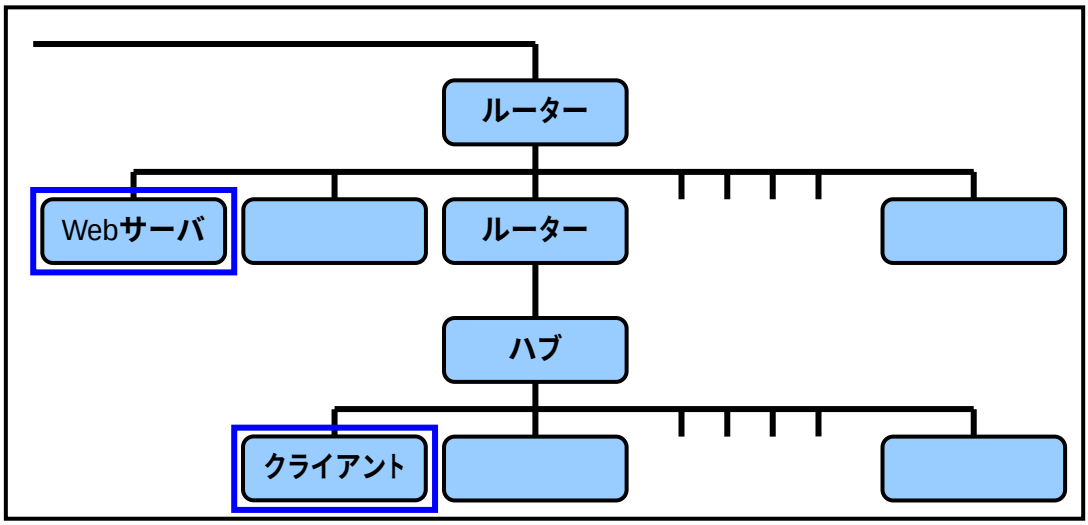


図 4.1　ネットワーク図

実験は、Web ページが置いてある PC 内でのアクセス、ネットワークからのアクセスと 2 種類行い、それぞれの Web ページにおいて、50 回ずつ計測を行った。3.3 で述べた通り Web ページはキャッシュに残るため、手作業によりキャッシュファイルを削除している。しかし、動的な Web ページであるサーブレットで製作された Web ページはこのような作業は必要ない。だが、画像は静的な Web ページであるため、文字＋画像のページはキャッシュを削除した。

この章では、様々な角度から結果を比較していく。。

4.1　DataBase に接続しない Web ページ

この節では DataBase に接続しない Web ページを比較する。

4.1.1　Web ページのデータ量による比較

図 4.1 は、Web ページのデータ量と計測された時間の相関図である。

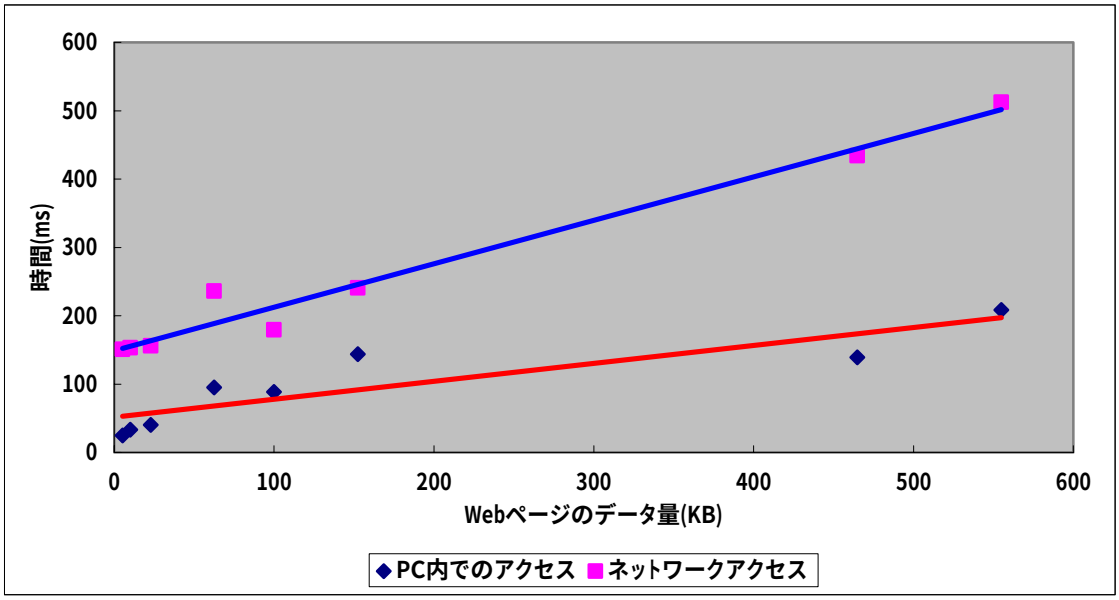


図 4.1　データ量と計測された時間の相関図

Web ページとして表示されるデータ量によって、表示に要する時間は比例しているといえる。

ここで、画像の表示ではなく、同サイズの文字だけの Web ページにしたらどうなるかと考え、画像のみ・文字＋画像の Web ページを文字のみで表現し計測してみたところ、画像を含む Web ページに要した時間よりも、わずかだが、少ない結果になった。これは、ブラウザの特性の１つが影響していると考えられる。

そもそも Web ページが表示されるのには、まず、文字の部分だけを読み込み、その中で HTML のタグを探し出し、タグ機能を実行している。画像のように<IMG SRC=”URL”>を見つけ出すと、また新たにその URL 先の画像を読み込むようになっている。そのため、同サイズの文字のみの Web ページと文字＋画像のページに差ができたといえる。

4.1.2　接続形態による比較

図 4.2 は、DataBase に接続しない Web ページにおける、PC 内でのアクセスとネットワークアクセスの表示に要した時間の差をグラフにしたものである。

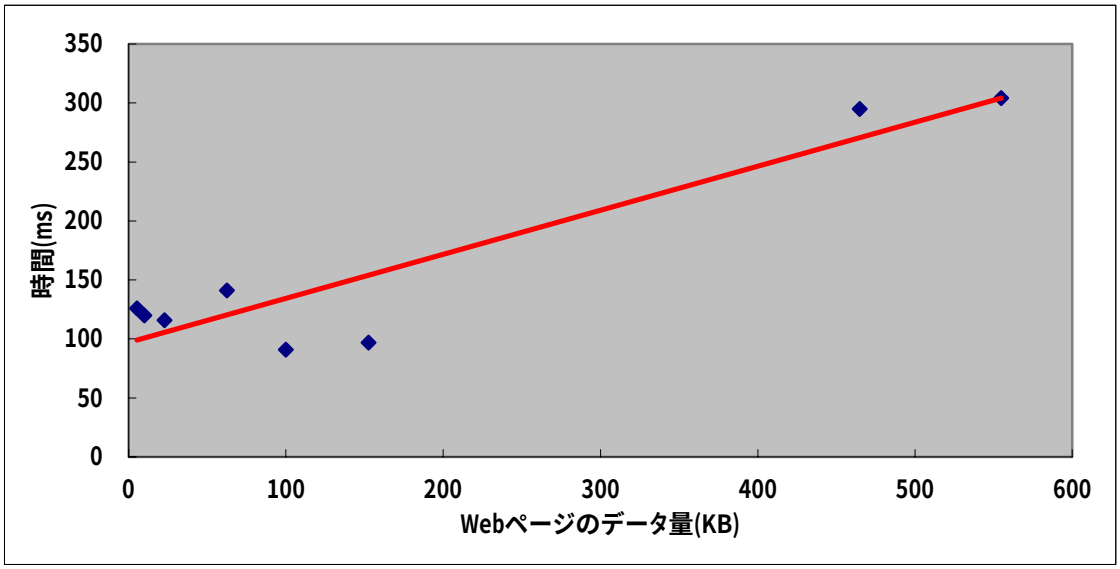


図 4.2　PC 内でのアクセスとネットワークアクセスの差

いずれも、ネットワークアクセスの方が PC 内でのアクセスより時間を要することが見て取れる。ネットワークアクセスと PC 内でのアクセスの違いは、データの転送だけである。ネットワークアクセスの方が時間を要するということは、ネットワークアクセスでは PC 内でのアクセスに比べ、データ転送に時間を要するということである。

しかし、この近似直線に当てはまらない点がある。これは実験の際に、なんらかのネットワークの負荷が生じたためであると考えられる。

4.1.3　記述言語による比較

DataBase に接続しない Web ページのそれぞれを、HTML 言語と Java 言語により記述した。それぞれのページの計測結果は表 4.1・表 4.2 になった。

表 4.1　PC 内でのアクセスの Web ページの計測結果（単位 ms）

	HTML	Java
文字少	33	35
文字少画像小	94	97
文字少画像大	139	147

表 4.2　ネットワークアクセスの Web ページの計測結果（単位 ms）

	HTML	Java
文字少	153	155
文字少画像小	236	256
文字少画像大	434	441

HTML 言語で記述している Web ページは、Web サーバからデータを受け取り、ブラウザによりタグを読み込み処理をする。それに対し、Java 言語で記述された Java Servlet は、クライアントからリクエストがあると Web サーバで処理を行い、HTML 言語でクライアントへ返す。つまり、Java 言語による Web ページは Web サーバでの処理の時間だけ時間が要することが考えられるが、今回作成した Web ページぐらいの処理であれば、表示に時間を要さないことがわかる。

4.2　DataBase に接続する Web ページ

この節では DataBase に接続する Web ページについて考察する。特に、DataBase に格納されているデータ量による比較を行った。

4.2.1　単一テーブルへの操作による比較

ここでは、画像の表示のない文字のみで構成される Web ページで比較する。

これら Web ページは、ある 1 つのテーブルを検索し、すべてのデータを表示するようにしている。本研究では DataBase に格納されているデータ量を 3 種類用意しているが、文字少の Web ページは、DataBase のデータ量が変わっても、テーブル内のデータ量が変わらないようにしている。文字多の Web ページでは、DataBase のデータ量が変われば、テーブル内のデータ数が、50 行、500 行、5000 行のように変化する。表 4.3・表 4.4 が計測結果である。

表 4.3　文字少の Web ページの計測結果（単位 ms）

DB 内のデータ量	多	中	少
PC 内でのアクセス	58	57	58
ネットワークアクセス	159	157	158

表 4.4　文字多の Web ページの計測結果（単位 ms）

DB 内のデータ量	多	中	少
PC 内でのアクセス	4592	495	84
ネットワークアクセス	3576	421	161

文字少の Web ページは、DataBase に格納されているデータ量により、変化がなかった。よって、ある 1 つのテーブルのデータ量が変わらない場合、そのテーブルだけの操作であれば検索から表示までに要する時間は変わらないことが考えられる。

文字多の Web ページでは、DataBase に格納されているデータ量により時間が異なった。これは、検索に時間を要した点と、データを返すのに時間を要した点の 2 点が考えられる。そのため、500 行、5000 行の表示をさせるものから、格納されているデータ量が少ないものと同じ、50 行の表示をする Web ページを作成し、計測してみたところ、表 4.5 のような結果が得られた。

表 4.5　文字多の Web ページの計測結果（単位 ms）

DB 内のデータ量	多	中	少
PC 内でのアクセス	84	85	58
ネットワークアクセス	163	160	161

結果、DataBase に格納されているデータ量が少ないときのものと、ほぼ同じ時間を要したことが確認できた。よって、テーブル内のデータ量に変化があっても表示するデータ量が同じであれば、Web ページが表示されるまでの時間はあまり変化しないことが言える。

4.2.2　複数テーブルへの操作による比較

ここでは、文字＋画像の Web ページで比較する。

文字＋画像の Web ページは、複数のテーブルを検索し、一部のデータのみを表示するようにしている。これらの Web ページで、文字の多・少の違いは、検索するテーブル数、また SQL 文により得られるデータ量である。また画像の大小の違いは、それぞれの画像の大きさである。表 4.6・表 4.7 が計測結果である。

表 4.6　PC 内でのアクセスの Web ページの計測結果（単位 ms）

DB 内のデータ量	多	中	少
文字少画像小	207	98	82
文字少画像大	207	87	79
文字多画像小	269	218	221
文字多画像大	270	212	212

表 4.7　ネットワークアクセスの Web ページの計測結果（単位 ms）

DB 内のデータ量	多	中	少
文字少画像小	774	566	545
文字少画像大	739	630	640
文字多画像小	425	382	378
文字多画像大	531	503	469

ここで、それぞれの Web ページが検索するテーブル内のデータ量について考察してみる。文字少＋画像の Web ページは、customer、item、orders、order_item の 4 つのテーブルを、文字大＋画像の Web ページは item、maker の 2 つのテーブルに対し操作を行う。そ

れに伴う、検索の量の違いは表 4.8 のようになる。

表 4.8　検索対象のテーブル内のデータ量（単位 行）

DataBase 内のデータ量		多	中	少
文字少画像	customer	5000	500	50
	item	20000	2000	300
	orders	200	200	20
	order_item	6050	616	62
文字多画像	item	20000	2000	300
	maker	60	60	60

文字少＋画像の Web ページは、データ量中は少の約 6600 倍・データ量多は少の約 6500 万倍にも及ぶ。また文字多画像の Web ページでは、データ量中は少の約 6.6 倍・データ量多は少の約 66 倍である。それだけの検索量にもかかわらず、計測結果はあまり差がなかった。つまり、DataBase に格納されているデータ量には関係がないと考えられる。

DataBase は、複数のテーブルを結合して扱うことができる。上記の実行では結合を行っていない。そのため、テーブルを結合させる JOIN という SQL コマンドを使用した Web ページを作成し比較を行った。実際に使用した SQL 文は以下のようになる。

SELECT i_title, i_subject, oi_num, i_price, i_image FROM (order_item INNER JOIN orders ON order_item.oi_o_id = orders.o_id) INNER JOIN item ON order_item.oi_i_id = item.i_id WHERE orders.o_c_id = ?

結合したものは order_item テーブルと orders テーブル、その結合されたテーブルと item テーブルである。それぞれ結合時の関係はそれぞれ order_item.oi_o_id = orders.o_id と order_item.oi_i_id = item.i_id にしている。これは RDBMS の本来の考え方である、1 つのデータは 1 つのテーブルで管理し、データの整合性を保障考え方に準拠したものである。

表 4.9 がその結果である。データ量により多少の違いはあったが、目立つほどの差はみられなかった。

表 4.9　JOIN の使用あり／なしでの Web ページの計測結果（単位 ms）

	ネットワークアクセス		PC 内でのアクセス	
JOIN の使用	あり	なし	あり	なし
多	750	774	207	207
中	619	566	89	98
少	658	545	76	82

第５章　おわりに

本研究では、様々な Web ページを作成し、その表示に要する時間を計測した。その結果、作成した Web ページでは以下の要因により、表示に要する時間は変動することがわかった。

・Web ページとして表示されるデータ量

・Web ページへのアクセス形態

しかし、DataBase に格納されているデータ量にはあまり影響しないこともわかった。

しかしながら、今回は表示にかかる時間を計測しただけであるので、どの部分に大きな負荷が生じたのかを確認することができなかった。今後は Web サーバにもモニタを設置し、リクエストからレスポンスに要する時間や、DataBase への接続、処理に要する時間等を計測することが、課題としてあげられる。

また、人気サイトでは複数のアクセスが生じる。その場合どうなるかといったシミュレーションも行う必要がある。

Web ページの製作の上でも以下のような課題があげられる。DataBase と接続した Web ページに使用した SQL コマンドは、SELECT のみである。しかし、SQL コマンドには、ほかにも INSERT（挿入）・DELETE（削除）・UPDATE（修正）等がある。SELECT は検索するだけなので、実際に DataBase に対して、データの操作を行う他のコマンドを使用する必要がある。また、使用した画像は DataBase には関係ないところに置いてある。DataBase には LOB 型という、大きなデータを格納することができるデータ型がある。これについても比較する必要がある。

謝辞

本研究にあたり、最後まで熱心な御指導をいただきました田中章司郎教授には、心より御礼申し上げます。また、田中研究室生の貫目洋一君、高木明君、中村吉郎君、長田昌訓君、喜代吉容大君、鷲見明君、垰田千帆さん、田辺知里さん、堀隆志君には、本研究に関して数々の御協力と御助言をいただきました。厚く御礼申し上げます。

なお、本論文・本研究で作成したプログラム及びデータ、並びに関連する発表資料等のすべての知的財産権を、本研究の指導教官である田中章司郎教授に譲渡致します。

参考文献・URL

[1] インターネット接続サービスの利用者の推移
http://soumu.go.jp/s-news/2002/021227_1.html

[2] 石井達夫：改定第 3 版 PC UNIX ユーザのための PostgreSQL 完全攻略ガイド、株式会社 技術評論社、2001

[3] MSDN Online Japan
http://www.microsoft.com/japan/msdn/default.asp