〔非公開〕

ＴＲ－Ｍ－００２３

効果音データベースの構築

森崎　修司　Shuji MORISAKI（奈良先端大）
井上　誠喜　Seiki INOUE

１９９７．８．１

ＡＴＲ知能映像通信研究所

1 まえがき

記憶装置の大容量化, 計算機の高速化により大量の静止画像, 動画像, 音声などのデータを扱うことが可能となった. しかし, これらの異なる種類のデータに対しそれぞれのデータベースを構築し, 運用, 管理することは効率的ではない. これら静止画像, 動画像, 音声などの異なる種類の大量のデータを統合的に扱うことのできるデータベースとしてマルチメディアデータベースがある. 今後, このようなマルチメディアデータベースの必要性は高まっていくものと考えられる. そこで, 本実習ではマルチメディアデータベースとの親和性が高いオブジェクト指向データベースを用いて, データベースを設計し, 実際に効果音データを蓄積し, web ブラウザを使い, それらを検索し検索結果から希望の効果音を再生できるシステムを構築した.
今回, 扱ったデータは効果音データのみであるが, 異なる種類のデータを蓄積, 検索することも容易である.

2 システムの概要

システムはデータベースとインタフェースで構成される. インタフェースはデータベースのコマンドを必要に応じて呼び出す. データベースサーバと http サーバは同一のサーバである. インタフェースによりユーザからの検索の条件を受け取り, 条件に応じたデータベースのコマンドを呼び出す. データベースからの出力をインタフェースによりヘッダを付加し表示する.

2.1 データベース

オブジェクト指向データベースの管理システムとして object store を利用し, データの管理を行う. object store は C++ 言語のライブラリである. また, 従来のデータベース管理システムでは扱えない非レコード型のデータを扱うことができる. また, スキーマ変化の機能を使うことにより, 将来データベースのスキーマに変更が加えられても, それまでに蓄積されたデータをそのまま流用することができる. データベースの主要な部分は C++ で記述している.

各々のデータは時間長, データが保存してあるファイルのパスなどの属性を持つ. これらの属性の管理方法は, 次の点に留意して行った.

- 再利用が容易なこと
  新たに異なる種類のデータを蓄積する場合でもデータベースの主幹となる部分の変更がなるべく少なくなるようにする. そのため, 属性に階層構造を持たせ, 異なる種類のデータであっても, 共通にもつ属性の管理は共通の方法で行う.

- 多様なデータを扱えること
  多様なデータを蓄積できるように, それぞれの階層の属性にそれ独自の管理方法を持たせる.

- 属性の表示形式の統一
  それぞれの階層構造がもつ属性は, その階層でのみアクセスできそれぞれ独自の画面へ表示する等のメソッドを持つ. しかし, これでは統一した表示形式でそれぞれの属性をあらわすことが難しくなる. そこで, ある階層において表示形式を定義し, 属性を表示するためのメソッドを定義する. 表示はすべてその階層を通じて行うことで統一した表示方法を実現する. 表示形式を定義している階層を、別に用意した表示形式を定義している階層に再定義するだけで, 属性の表示形式の変更が行えるようにする.

- 柔軟性
  データベース中のそれぞれのデータに対し, それぞれを再生するメソッドを持たせる. データの再現はそのメソッドを呼び出すことで行うため, 異なる再現方法を持つデータに対しても共通のメソッドを通じて再現が可能である.

2.2 インタフェース

ユーザインタフェース部分は web ブラウザを使用した, ネットワークを介するためデータベースへの問い合わせは場所を選ばずに行うことができる. web ブラウザの特性上, ユーザとのやりとりはクライアントの位置にかかわらず透過的に行える. インタフェースはシェルスクリプトと HTML により記述している. また, インタフェースを制御するのコードの軽量化を計り, インタフェースの変更を行う場合に, 大規模な変更が必要にならないように配慮した.

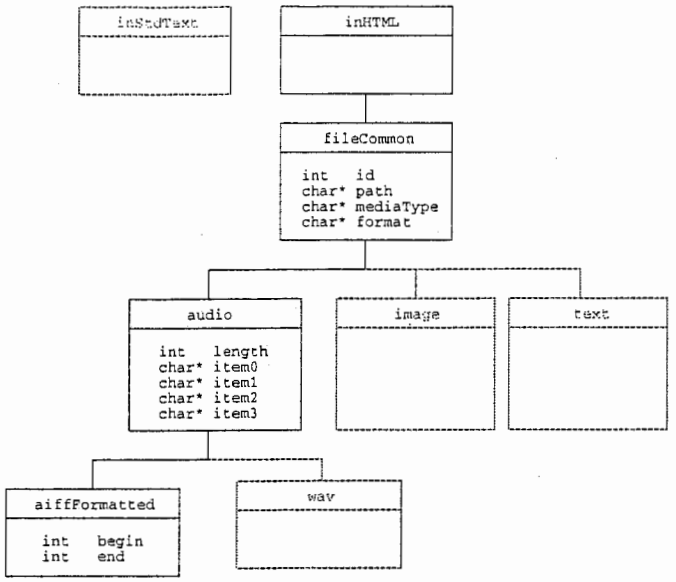

図 1: 図 1 データベーススキーマのクラス階層

3 効果音データベース

3.1 データの階層構造

本実習で作成したデータベースでは aiff 形式の効果音を扱う．この効果音のデータの構造を図 2 に示す．階層と階層がもつ属性を表にしたものが表 1 である．

また, 以下はそれぞれのクラス階層がもつ属性である．

1. inHTML
   本実習で作成したデータベースへの問い合わせは, 画面への表示はすべて web ブラウザを通して行う．そのため, まず最下層に html 形式の表示形式を定義するこのクラスを用意する．

2. CfileCommon
   表示形式を定義するクラス inHTML から派生するクラスである．データベース中のデータがすべて共通にもつ抽象クラスである．データベース中のデータはすべてこの CfileCommon クラスとから派生していなければならない．このクラスでは実際のデータが保存されているファイルのパス, 通し番号, データの種類, データのフォーマットを属性としてデータメンバに持つ．

3. Caudio
   クラス CfileCommon から派生する．データベース中の音声データがすべて共通にもつ抽象クラスである．音声データのクラスはすべてこのクラスから派生していなければならない．このクラスでは, 音声の時間長, その音声に関する説明として 4 つの項目を属性としてデータメンバに持つ．

4. Caiff
   クラス Caudio から派生する．データベース中の aiff 形式のフォーマットを持つ音声データはこのクラスのインスタンスであるか, このクラスから派生されるクラスのインスタンスでなければならない．このクラスは属性としてデータメンバに開始時刻と終了時刻を持つ．本実習で実際に蓄積した効果音データはこのクラスのインスタンスである．

クラス名	属性
inHTML	なし
CfileCommon	ID, パス, データの種類
Caudio	時間長, 音の説明 (4 項目)
Caiff	開始時刻, 終了時刻

表 1

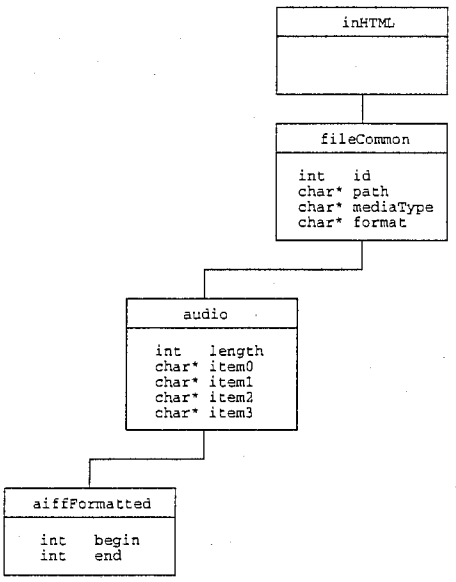

図 2: 図 2 効果音データベースのクラス階層

3.2 データベースの操作

データベースを操作するために以下の 5 つのコマンドを作成した.

1. dbinit
   空のデータベースを作成し, 初期化する. このコマンドは web ブラウザからは実行できない.

2. add
   データベースに指定されたファイルの内容を追加する. このコマンドは web ブラウザからは実行できない.

3. show
   データベース中のすべてのデータを表示する. このコマンドは web ブラウザから実行できる.

4. id
   データベース中のすべてのデータのうち指定された範囲の ID をもつデータを表示する. このコマンドは web ブラウザから実行できる.

5. item
   データベース中のすべてのデータのうち指定された項目に検索文字列を含み, 指定された範囲の ID 番号をもつデータを表示する. ID 番号の範囲は省略できる. このコマンド web はブラウザから実行できる.

6. perform
   指定された ID を持つオブジェクトに対して, 再生のメソッドを呼び出す. 再生のメソッドはそれぞれのオブジェクトごとに定義されている

3.3 インタフェースの処理

インタフェースではユーザから検索条件を受け取り, それに応じて検索を行うためのデータベースを操作するコマンドを呼出し, 結果を表示する. インタフェースには HTML 形式でかかれたページとクライアントから送られてくる検索条件を処理するサーバ側のシェルスクリプトが 3 つある. また, HTML 形式のページはファイルから読み込みスクリプトで生成する.

1. menu
   検索条件を指定するページを生成するスクリプトである. このスクリプトは利用可能なデータベースを検索し, それを定型のページに挿入して検索条件を指定するページを生成する.

2. scriPt
   ユーザが指定した検索条件を受け取り，結果を表示するスクリプトである．ユーザが指定した検索条件に応じて，データベースを操作するコマンドに検索条件にあわせた引数を渡す．このコマンドから出力された結果にHTMLのヘッダを付加し，検索結果のページを生成しクライアント側に送り出す．

3. play
   ユーザが再生を要求したデータを再生する．ユーザの指定を受け取り，それをそのままデータベースを操作するコマンドに渡す．

4 効果音データベースへの問い合わせ

実際のデータベースへの問い合わせの処理を以下に述べる．

4.1 データベースの初期化

まず，問い合わせの対象となるデータベースを作成する．データベースを初期化するコマンドdbinitをコマンドラインから実行し，データベースを初期化する．次に，データベースにデータを蓄積する．これは実際の効果音データではなく，そのデータに関する属性を蓄積する．実際の効果音のデータをデータベースに蓄積することも可能であるが，本実習では別の記憶装置上のファイルに保存してあるものを使用する．属性が記述されているファイルと初期化したデータベースを指定しコマンドaddをコマンドラインから実行する．以上を行った上で効果音データベースへの問い合わせを実行する．なお，データベースの初期化はすべてUNIXのコマンドラインから実行する．

4.2 検索条件の指定

クライアントから検索条件を入力するページへのリクエストが送られてくると，サーバのゲートウェイスクリプトmenuは検索条件を指定するページに，その時点で利用可能なデータベース名を挿入して送り出す．

検索条件はクライアントのHTML形式のページに入力する．検索するデータベース名を指定していない等の明らかに誤った入力を検出した場合には，クライアント側で，その誤りに対して警告を出す．検索条件がクライアントから送られてくるとゲートウェイスクリプトscriPtは検索条件に応じたデータベース操作コマンドを呼び出す．そのあと，検索結果をデータベース操作コマンドから受け取り，HTML形式のヘッダを付加し，クライアント側に送り出す．

クライアント側から再生のリクエストを受けた場合にはゲートウェイスクリプトplayが対象となるデータのIDを受け取りデータベースを操作するコマンドperformを呼び出す．コマンドperformでは，指定されたIDを持つオブジェクトを検索し，そのオブジェクトが持つ再生のメソッドを呼び出す．

4.3 効果音データベースの再生方法

本実習で作成したデータベースに蓄積されているaiff形式の効果音データのオブジェクトは，次のような再生メソッドにより効果音を再生する．効果音データはデータベースサーバ上ではなく，別のサーバ上に保存してある．また効果音を再生するコマンドplayaifc2が効果音を保存してあるサーバ上にある．データベースサーバからこのコマンドをリモートシェルを使用して呼び出し，効果音を再生する．

5 まとめ

本実習ではマルチメディアデータベースの第一歩として効果音データベースを作成した．まず，様々なデータを蓄積できるデータベースの設計を行い，実際に効果音のデータが持つ属性を蓄積した．そのデータにwebブラウザを用いて検索を行い，指定した効果音を再生できるシステムを作成した．
その結果，データベース中のオブジェクト毎に再生用のメソッドを用意することより再生方法に依存しないデータベースの構築が可能となった．また，データベース中のすべてのオブジェクトが共通のクラスから派生させることで，インタフェース部分の変更，表示形式の変更が容易にできることがわかった．
今後の課題
今回，私の力量不足がかなり影響して，当初の目標に達することがほとんどできてない．これはライブラリobject storeが，私にとって非常に扱いづらいものであったためである．当初の目標は，データベース内部にトリー構造をも

つデータベースを作成することであったが, 実現はされていない. また, 蓄積されたデータ量が増えると, サーバとクライアント間に容量の大きいコネクションが確保されない限り, web ブラウザでの検索は困難になると考えられる. エントリが 2000 を越えると結果の表示に数分かかった. データベースが大規模になればインタフェース部分の変更が必要になる. インタフェース部分のコードの軽量化を意識してシステムを設計したが, 実際に, インタフェースを変更しその作業量についての考察も行う必要がある.

インタフェース部分のプログラム

インタフェース部分のプログラムは次の3つです．インタフェース部分のプログラムCGIにより呼び出されます．すべてシェルスクリプトで記述しています．簡単な処理の説明を述べてからソースを示します．

1. menu

   データベースを検索する条件を入力するページを生成します．ページ全体を前半と後半にわけそれらを別々のファイルに保存しています．前半と後半の間に「利用可能なデータベース」として次のパスを参照して"ls" の結果を表示しています．

   ```
   /usr2/www/cern_httpd3.0/public_html/morisaki/database/
   ```

   ページの前半部分のファイルは

   ```
   /usr2/www/cern_httpd3.0/public_html/morisaki/webDoc/former2.html
   ```

   後半部分のファイルは

   ```
   /usr2/www/cern_httpd3.0/public_html/morisaki/webDoc/latter2.html
   ```

   です．

2. scriPt

   指定された条件を持つエントリをデータベースから検索するためにデータベース部分のコマンドを実行し，結果のページを生成します．タイトルなどのヘッダ部分に相当する部分を

   ```
   /usr2/www/cern_httpd3.0/public_html/morisaki/webDoc/header2
   ```

   にフッタ部分を

   ```
   /usr2/www/cern_httpd3.0/public_html/morisaki/webDoc/footer2
   ```

   に記述してあります．このスクリプトでは，入力されたデータベース名，選択された検索方法，入力されたID番号，入力された検索文字列，選択された検索対象項目を検索方法に応じてデータベース部分のコマンドに渡します．具体的には以下のようになっています．

選択された検索方法	コマンド	コマンドに渡す引数
データベース中のエントリをすべて表示	show	データベース名
ID番号による検索	id	データベース名 検索開始ID 検索終了ID
文字列による検索	item	データベース名 検索文字列 項目1 項目2 項目3 項目4
ID番号と文字列による検索	item	データベース名 検索文字列 項目1 項目2 項目3 項目4 検索開始ID 検索終了ID

3. perform

   検索の結果のページで指定された項目の音を出し，出しおわったら再び検索の結果のページへ戻ります．

ソース

ゲートウェイスクリプトでは標準出力への出力はクライアントへの出力になります．(1行すべてがコメントのときには，そのコメントはコメントの下の行のスクリプトに対するものです．)

1. menu


```
#!/bin/sh

FILESROOT=/usr2/www/cern_httpd_3.0/public_html/morisaki/
# 利用可能なデータベース名を一時的にこのファイルに書き出します
TMP="$FILESROOT"/tmp/dirList
THE_FORMER="$FILESROOT"/webDoc/former2.html # 前半部分
THE_LATTER="$FILESROOT"/webDoc/latter2.html # 後半部分

DBROOT="$FILESROOT"/database/ #データベースディレクトリのパスです

# HTML形式のためのヘッダです．2行目の空白は必ず書かなければなりません．
echo Content-Type: text/html
echo

cat $THE_FORMER # ページの前半部分を出力します．

ls $DBROOT > $TMP # データベースディレクトリの内容をTMPにいったん書き出します．

echo '<table cellpadding=1 width=80% cellspacing=5>'
echo '<tr><td align=center>利用可能なデータベース</td>'
echo '<td><font color=black>'
cat $TMP  # データベースディレクトリの内容を標準出力に出力します．
echo '</font></td></tr>'
echo '<tr>'
echo '<td align=center>検索するデータベース名'
echo '</td>'
echo '<td><input type="text" name="dbname" size=20></td>'
echo '</tr>'
echo '</table>'

cat $THE_LATTER # ページの後半部分を出力します．
```

このスクリプトにより、生成されるHTMLは以下のようになっています。

＜前半部分＞

java scriptで明らかに誤った入力に対し、警告を出します。また、検索条件を送る(sumbitする)前に検索条件の確認をします。簡単なスクリプトなので、見てもらえればわかると思います。最後の方にHTMLタグに

```
<form METHOD="POST" ACTION="/cgi-bin/scriPt" onsubmit="return checkQuery()">
```

という部分があります。ここで、ACTION="/cgi-bin/scriPt"の部分で検索条件を送ったときに呼び出されるゲートウェイスクリプトを指定しています。

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML VERSION="2.0">
<HEAD>
<!-- WEBMAGIC VERSION NUMBER="2.0.1" -->
<!-- WEBMAGIC TRANSLATION NAME="ServerRoot" SRC="/var/www/htdocs/" DST="/" -->
<!-- WEBMAGIC TRANSLATION NAME="ProjectRoot" SRC="./" DST="" -->
<TITLE>test </TITLE>
<script language="JavaScript">
```

```
function checkQuery(){
// 上から数えたformの順番 typeは検索方法 chechedは検索対象となる項目
   var dbName      = 0;
   var typeAll     = 1;
   var typeID      = 2;
   var typeStr     = 3;
   var typeStrID   = 4;

   var checked1    = 5;
   var String      = 6;
   var checked2    = 7;
   var checked3    = 8;
   var checked4    = 9;


   var IDFrom      = 10;
   var IDTo        = 11;

   // データベース名が入力されてないとき
   if (document.forms[0].elements[dbName].value==""){
      alert("database name must be specified.");
      return false;}

   // 「すべてを表示する」が選ばれたときの処理。確認をとります
   // うっとおしいときはコメントアウトして下さい
   if (document.forms[0].elements[typeAll].checked==true){
      return confirm("will show all elements.");}

   // 「IDによる検索」が選ばれたとき
   // 番号が入ってないとき、開始が終了より大きい数のときは警告を出します
   if (document.forms[0].elements[typeID].checked==true){
      var from = document.forms[0].elements[IDFrom].value;
      var to   = document.forms[0].elements[IDTo].value;

      if ( (from == "") || (to == "") ){
         alert("ID numbers must be specified.");
         return false;}
      if (from > to){
         alert("ID numbers are wrong.");
         return false;}

   return confirm("will show elements with ID "+from+" to "+to);}

   // 「文字列による検索」が選ばれたとき
   // 文字列が入ってないときやどの項目も選ばれてないときに警告を出します
   if (document.forms[0].elements[typeStr].checked==true){
      if (document.forms[0].elements[String].value==""){
         alert("String searched for must be specified.");
         return false;}
      if ( (document.forms[0].elements[checked1].checked==false) &&
           (document.forms[0].elements[checked2].checked==false) &&
           (document.forms[0].elements[checked3].checked==false) &&
           (document.forms[0].elements[checked4].checked==false) ){
         alert("No item selected. At least one item must be selected.");
         return false;}
   return confirm("will show elements including string '"
                  +document.forms[0].elements[String].value+"'.");}

   // 「IDと文字列による検索」が選ばれたときには上の「文字列検索」と
   // 「ID検索」の条件を1つでも満たしてないとき警告を出します
   if (document.forms[0].elements[typeStrID].checked==true){
      var from = document.forms[0].elements[IDFrom].value;
      var to   = document.forms[0].elements[IDTo].value;

      if ( (from == "") || (to == "") ){
         alert("ID numbers must be specified.");
         return false;}
      if (from > to){
         alert("ID numbers are wrong.");
         return false;}
      if (document.forms[0].elements[String].value==""){
         alert("String searched for must be specified.");
         return false;}
      if ( (document.forms[0].elements[checked1].checked==false) &&
           (document.forms[0].elements[checked2].checked==false) &&
           (document.forms[0].elements[checked3].checked==false) &&
           (document.forms[0].elements[checked4].checked==false) ){
         alert("No item selected. At least one item must be selected.");
         return false;}

      return confirm("will show elements including string '"
                  +document.forms[0].elements[String].value+"' with ID "
                  +from+" to "+to);}

}

</script>
</HEAD>

<BODY   BGCOLOR = "#aaaaaa"
        TEXT    = white
        LINK    = blue
        VLINK   = red >

<form METHOD="POST" ACTION="/cgi-bin/scriPt"
onsubmit="return checkQuery()">
```


＜後半部分＞

以下が後半部分です。前半と後半の間にデータベースのディレクトリに対し"ls"を実行してその結果を出力します。これは /usr2/www/cern_httpd_3.0/public_html/morisaki/webDoc/latter2.html に書かれています。<input> タグにより、ユーザからの入力を受け付けます。さらに、type=で入力部分をラジオボタン、チェックボックス、エディットなどから選び、NAME=でゲートウェイスクリプトに名前をつけて渡します。これ

を /usr2/www/cern_httpd_3.0/bin/cgiparse -init を実行し、 /usr2/www/cern_httpd_3.0/bin/cgiparse -form を実行することで、シェル変数 FORM_（NAME で指定した変数）に代入してくれます。たとえば、NAME=”foo” と <input> タグで指定しておけばシェル変数 FORM_foo に入力された値が代入されます。

```
<hr size=4 width=80%>

<!------------------------- 検索方法 ------------------------->

<table cellpadding=3 cellspacing=1 width=95% >
<tr>
<td rowspan=4 align=left> 検索方法 </td>
<td><input type="RADIO"  NAME ="type" VALUE="all" checked>
データベース中のエントリをすべて表示 </td>
</tr>

<tr>
<td><input type="RADIO"  NAME ="type" VALUE="id">
ID 番号による検索 </td>
</tr>

<tr>
<td><input type="RADIO"  NAME ="type" VALUE="string">
文字列による検索 </td>
</tr>

<tr>
<td><input type="RADIO"  NAME ="type" VALUE="stringID">
ID 番号と文字列による検索 </td>
</tr>

</table>
<hr size=4 width=80%>

<!--------------------- 文字列検索のための設定 ------------------->

<table width=95%>

<tr>
<td align=left rowspan=5> 文字列の指定 </td>
<td align=center><input type="checkbox" name="item1" checked> 項目 1 </td>
<td rowspan=5> の中に文字列 <input type="text" name="term" size=10> を含む
エントリ </td>
</tr>
<tr>
<td align=center><input type="checkbox" name="item2" checked> 項目 2 </td>
</tr>
<tr>
<td align=center><input type="checkbox" name="item3" checked> 項目 3 </td>
</tr>
<tr>
<td align=center><input type="checkbox" name="item4" checked> 項目 4 </td>
</tr>
</table>
<hr size=4 width=80%>

<!---------------------- id 検索のための設定 -------------------->

<table cellpadding=5 cellspacing=5 width=95%>

<tr>
<td align=left>ID の指定 </td>
<td align=center>ID が <input type="text" name="from" size=5> から
ID が <input type="text" name="to" size=5> までのエントリ </td>
</tr>
</table>

<hr size=4 width=80%>

<input type="SUBMIT" value="send">
<input type="RESET" value="reset">

</center>
</form>
</BODY>
</HTML>
```

2. scriPt

上の HTML のページで入力された内容を、このスクリプトで処理します。そしてデータベースに対するコマンド show、item, id のいずれかに条件を指定して渡し、結果に HTML のヘッダとフッタをつけて出力します。ヘッダ部分は /usr2/www/cern_httpd_3.0/public_html/morisaki/webDoc/header2 に、フッタ部分は /usr2/www/ に書いてあります。クライアントからの検索条件を処理する手順は以下のとおりです。クライアントから送られてくるデータは最初はひとつながりになっているのでその中からデータを 1 つ 1 つ切り出さなければいけません．ここでは cgiparse という cern の http サーバに付属してくるスクリプトを利用してクライアントから送られてくるデータを切り出しています．クライアントから送られてくるデータ名とその内容との対応を以下に示します．

データ名	値	内容
FORM_dbname		検索するデータベース名
FORM_type		検索方法
	all	すべてのエントリを表示
	id	ID 番号による検索
	string	文字列による検索
	stringID	文字列と ID 番号による検索
FORM_item1		文字列による検索を行うとき項目 1 を検索対象にするとき'on'
FORM_item2		項目 2 を検索対象にするとき'on'
FORM_item3		項目 3 を検索対象にするとき'on'
FORM_item4		項目 4 を検索対象にするとき'on'
FORM_term		検索文字列
FORM_from		検索開始 ID
FORM_to		検索終了 ID

以下がソースです．

(1行すべてがコメントのときには，そのコメントはコメントの下の行のスクリプトに対するものです．)

```
#!/bin/sh
# cgiparse を通すと <input name=''...'' は FORM... になります
#
# 例えば HTML で <input TYPE="TEXT" NAME="foo"> であれば cgiparse を通すと
# 入力された値が FORM_foo に入ります

CGIPARSE=/usr2/www/cern_httpd_3.0/bin/cgiparse # cgiparse のパス
FILESROOT=/usr2/www/cern_httpd_3.0/public_html/morisaki
HEADER="$FILESROOT"/webDoc/header2 # 結果のページの前半部分
FOOTER="$FILESROOT"/webDoc/footer2 # 結果のページの後半部分

# 検索結果を後で参照できるようにファイルに書き出しておきます
OUT="$FILESROOT"/tmp/result.html
DBROOT="$FILESROOT"/bin

eval '$CGIPARSE -init'  # cgiparse の実行 データを切り出します
eval '$CGIPARSE -form'  # cgiparse の実行 環境変数 FORM_... に内容が入ります


# 以下で文字列による検索のとき，どの項目を指定されたか調べます
if [ "$FORM_item1" = "on" ]; then ITEM1="true" # 項目1のチェックボックス
else ITEM1="false"
fi

if [ "$FORM_item2" = "on" ]; then ITEM2="true" # 項目2のチェックボックス
else ITEM2="false"
fi

if [ "$FORM_item3" = "on" ]; then ITEM3="true" # 項目3のチェックボックス
else ITEM3="false"
fi

if [ "$FORM_item4" = "on" ]; then ITEM4="true" # 項目4のチェックボックス
else ITEM4="false"
fi


# 入力されたデータベースを対象とします．
DBNAME="$FILESROOT"/database/"$FORM_dbname"

# データベース中のすべての項目を表示します コマンドは show (database 名)
SHOW_ALL='$DBROOT/show $DBNAME'

# ID 番号による検索をします コマンドは id (database 名) (開始 ID) (終了 ID)
ID_SEARCH='$DBROOT/id $DBNAME $FORM_from $FORM_to'

# 文字列による検索をします コマンドは
# item (database 名) (検索文字列) (項目1 (true なら対象項目 それ以外は
#             非対象項目) (項目2 (同様)) (項目3 (同様)) (項目4 (同様))
```



```
STR_SEARCH='$DBROOT/item $DBNAME $FORM_term $ITEM1 $ITEM2 $ITEM3 $ITEM4'

# 文字列と ID 番号の AND 検索をします コマンドは文字列による検索と同じ itemInHTML
# を使います 文字列による検索と同様にして，最後に (開始 ID) (終了 ID) を
# 付け加えます つまり
# item (db 名) (文字列) (項目1)..(項目4) [開始 ID] [終了 ID] です
STR_AND_ID_SEARCH='$DBROOT/item $DBNAME $FORM_term $ITEM1 $ITEM2 $ITEM3 $ITEM4 $FORM_from $FORM_

echo Content-Type: text/html      # HTML のヘッダです
echo

cat $HEADER             # 前半部分を出力します

# 選ばれた検索方法によりデータベースを扱うコマンドを実行します
# 音をならし終わったあと再び検索結果が表示できるようにファイル OUT に
# 検索結果を出力しておきます
case "$FORM_type" in
"all")              # すべてのエントリを表示
echo '<h2 align=center> データベース中のすべての項目 </h2><br><br>'>$OUT
echo 'eval $SHOW_ALL'>>$OUT
cat $OUT
;;

"id")               #  指定された ID による検索
echo '<h2 align=center> ID による検索の結果 </h2><br>'>$OUT
        echo "<center>$FORM_from から $FORM_to まで </center><br><br>">>$OUT
echo 'eval $ID_SEARCH'>>$OUT
cat $OUT
;;

"string")           # 文字列による検索
echo '<h2 align=center> 文字列による検索の結果 </h2><br>'>$OUT
echo '<center> '>>$OUT
        echo "$FORM_term を含むエントリ </center><br><br>">>$OUT
echo 'eval $STR_SEARCH'>>$OUT
cat $OUT
;;

"stringID")         # 文字列と ID による検索
echo '<h2 align=center> 文字列と ID による検索の結果 </h2><br>'>$OUT
echo "<center> $FORM_term を含み ID が $FORM_from から ">>$OUT
        echo " $FORM_to までのエントリ </center><br><br>">>$OUT
echo 'eval $STR_AND_ID_SEARCH'>>$OUT
cat $OUT
;;
esac;

cat $FOOTER  # 後半部分の出力
```

＜ヘッダ＞

スクリプト scriPt で使用されているヘッダ部分は以下のようになっています。java script の部分で画面のフェードインを行っています。

```
<html>
<head>
<tilte>result </title>
<script language="JavaScript">

function makearray(n) {
    this.length = n;
    for(var i = 1; i <= n; i++)
        this[i] = 0;
    return this;
}
hexa = new makearray(16);
for(var i = 0; i < 10; i++)
    hexa[i] = i;
hexa[10]="a"; hexa[11]="b"; hexa[12]="c";
hexa[13]="d"; hexa[14]="e"; hexa[15]="f";
function hex(i) {
   if (i < 0)
        return "00";
    else if (255 < i)
        return "ff";
    else
        return "" + hexa[Math.floor(i/16)] + hexa[i%16];
}
function setbgColor(r, g, b) {
    var hr = hex(r); var hg = hex(g); var hb = hex(b);
    document.bgColor = "#"+hr+hg+hb;
}
function fade(sr, sg, sb, er, eg, eb, step) {
    for(var i = 0; i <= step; i++) {
        setbgColor(
        Math.floor(sr * ((step-i)/step) + er * (i/step)),
        Math.floor(sg * ((step-i)/step) + eg * (i/step)),
        Math.floor(sb * ((step-i)/step) + eb * (i/step)));
    }
}
/* Usage:
 *   fade(inr,ing,inb, outr,outg,outb, step);
 * example.
 *   fade(0,0,0, 255,255,255, 255);
 * fade from black to white with very slow speed.
 *   fade(255,0,0, 0,0,255, 50);
 *   fade(0xff,0x00,0x00, 0x00,0x00,0xff, 50); // same as above
 * fade from red to blue with fast speed.
 * step 2 is very fast and step 255 is very slow.
 */
function fadein() {
    fade(64,64,64, 255,255,255, 90);
}
function fadeout() {
    /*fade(0,0,0, 255,255,255, 120); */
```



```
}
/* do fadein */
fadein();

</script>

</head>
<body BGCOLOR = white TEXT = black LINK=blue VLINK=red>
<table border=3 cellspacing=3 cellpadding=2>
<tr><td align=center>ID</td>
<td align=center> 項目 1 </td>
<td align=center> 項目 2 </td>
<td align=center> 項目 3 </td>
<td align=center> 項目 4 </td>
</tr>
```

＜フッタ＞
スクリプト scriPt で使用されているヘッダ部分は以下のようになっています。<table> タグの終わりと <HTML> タグの終わりを書いているだけです。

```
</table>
<BR>
<hr size=6 width=60%>
<h1 align=center><a href="/cgi-bin/menu"> 戻る </a></h1>
<hr size=6 width=60%><br><br>
</body>
</html>
```

3. perform
ID 番号で指定された項目に対して再生のメソッドを呼び出します。ID 番号は web ページの play ボタンを押すことで、クライアントから送られてきます。

```
#!/bin/sh
# cgiparse を通すと FORM_... は FORM タグで指定された名前が... になります
#
# 例えば HTML で <input TYPE="TEXT" NAME="foo"> であれば cgiparse を通すと
# 入力された値が FORM_foo に入ります

CGIPARSE=/usr2/www/cern_httpd_3.0/bin/cgiparse #cgiparse のパス
MENU=/usr2/www/cern_httpd_3.0/cgi-bin/menu # menu にもどるためのパス
FILESROOT=/usr2/www/cern_httpd_3.0/public_html/morisaki
TMPDB="$FILESROOT"/tmp/dbname


TMP="$FILESROOT"/tmp/temp

HEADER="$FILESROOT"/webDoc/header3 # 結果のページの前半部分
TEMP_STD_OUT="$FILESROOT"/tmp/result.html # 結果のページの内容
FOOTER="$FILESROOT"/webDoc/footer2 # 結果のページの後半部分

eval `$CGIPARSE -init`
eval `$CGIPARSE -form`
```

```
echo "Content-Type: text/html" # HTML のヘッダ
echo

read DBNAME<$TMPDB      # 問い合わせを行なったデータベース名の取得
TMP=eval`$FILESROOT/bin/perform $FILESROOT/database/$DBNAME $FORM_id`

cat $HEADER         # HTML の前半部分
cat $TEMP_STD_OUT   # 再生の前に行なった検索結果をそのまま使う
cat $FOOTER         # HTML の後半部分
```

データベース部分

コマンドは以下の5つがあります．これらのプログラムのソースはすべて

```
/usr2/www/cern_httpd3.0/public_html/morisaki/source/
```

の中にあります．

1. dbinit （ファイル名）
   （ファイル名）で指定されたファイルをデータベースファイルとし，初期化します．web上で見る場合にはデータベースファイルのパーミッションを書き込み可にしておく必要があります． webでは「利用可能なデータベース」は次のパスを参照しています．

   ```
   /usr2/www/cern_httpd3.0/public_html/morisaki/database/
   ```

2. add （ファイル名1） （ファイル名2）
   （ファイル名1）で示されたファイルのリストを（ファイル名2）で指定されたデータベースファイルに加え，保持されているファイル数のカウンタを更新します．

3. show （ファイル名）
   （ファイル名）で指定されたデータベースのエントリをすべて表示します．

4. id （ファイル名） （検索開始番号） （検索終了番号）
   （ファイル名）で指定されたデータベースに保持されているエントリのID番号を検索し，（開始番号）から（終了番号）までのIDを持つエントリを表示します．

5. item （ファイル名） （検索文字列） （項目1） （項目2） （項目3） （項目4） [検索開始番号] [検索終了番号]
   （ファイル名）で指定されたデータベース中に保持されているエントリの中で検索文字列を含むエントリを表示します．（項目1）から（項目4）でその項目を検索対象とするかしないかが決められます．'i' が指定されたときには，その項目を検索対象とします．'i' 以外の文字が指定されたときにはその項目に（検索文字列）が含まれていても表示しません．（項目1）から（項目4）は省略できません．[検索開始番号] と [検索終了番号] が指定された場合には開始番号から終了番号までのIDを持つエントリを検索します．[検索開始番号] と [検索終了番号] が指定された場合には開始番号から終了番号までのIDを持つエントリを検索します．[検索開始番号] と [検索終了番号] が指定された場合には開始番号から終了番号までのIDを持つエントリを検索します．[検索開始番号] と [検索終了番号] は省略できます．省略した場合にはデータベース中のすべてのエントリが検索の対象となります．

6. perform (ファイル名) （ID番号）
   （ファイル名）で指定されたデータベース中に保持されているエントリの中で指定された（ID番号）を持つエントリを再生させます．

実はクラスの設計はまったくダメです．理由はデータベースに蓄積されている内容を扱う部分をすべてカプセル化しようとして，結局うまくできてないからです．あとから，templateを使えばうまくやれることがわかりましたが実際にはそこまでできていません．また，テキストの表示形式をデータベースのオブジェクトに持たせるのも正しいとはいえません．それぞれのコマンドshow, dbinit, add のそれぞれに持たせなければなりませんが，これらのコマンドのカプセル化に失敗したので，仮にデータベースのオブジェクトにテキストの表示形式のメソッドを持たせています．

ヘッダ，ソース中の関数名，変数名，クラス名はおよそ次のように分類できます．

- "_" で区切られているもの
  "_" で区切られているものは，ほとんどobject storeに関する変数名，クラス名です．たとえばget_os_typespec()などです．

- 区切りの後の文字が大文字になっているもの
  すべて，私が定義したものです．たとえばmediaTypeなどです．

- Cで始まるクラス名
  テキストの表示形式のクラスinHTMLと先頭がCで始まるクラス名をもつクラスは私が作成したクラスです．先頭がCで始まるクラスは先頭のCをとったヘッダファイルに書いています．inHTMLはbase.hというヘッダファイルにかいてあります．

以下にそれぞれのファイルに書いてあるプログラムについて説明します．プログラム中のコメントが行頭から始まっている場合には，そのコメントの下の行に対するコメントです．

- ヘッダヘッダはbase.h, fileCommon.h, audio.h, aiff.h, property.hの5つです．

  1. base.h
     テキスト表示形式を定義するクラスinHTMLを定義しています．

     ＜リスト＞

```
/**

基底クラスです． どのようなテキスト表示形式を用いるかを決めます．
inHTMLから派生させたクラスはHTML形式で表示できます．
このヘッダファイルで表示形式が変えられたくないものは
このクラスのメソッドを経由しないで，表示します．

*/
// c/c++ ヘッダ
#include <iostream.h>
// object store ヘッダ
#include <ostore/ostore.hh>

class inHTML {
 public:
  // object storeにこのクラスの構造を知らせるための関数
  // object storeのデータベースにアクセスする時には必ず
  // それぞれのクラス，型のos_typespecが必要です
  static os_typespec *get_os_typespec();

  void kaigyou(){                    // 改行のためのメソッドです
    cout << "</tr>\n<tr>";}
  void hyouji(char string[]){        // 引数で与えられる文字を表示します
    cout << "<td>" << string << "</td>";}
  void hyouji(int number){           // 引数で与えられる整数を表示します
    cout << "<td>" << number << "</td>";}
  void hyouji(double number){        // 引数で与えられる実数を表示します
    cout << "<td>" << number << "</td>";}

  // 以下の3つの関数は第1引数で与えられたものを第2引数で与えられる位置に
  // 表示します
  void hyouji(char string[], char align[]){
    cout << "<td align=" << align << ">" << string << "</td>";}
  void hyouji(int number, char align[]){
    cout << "<td align=" << align << ">" << number << "</td>";}
  void hyouji(double number, char align[]){
    cout << "<td align=" << align << ">" << number << "</td>";}
```

```
  void tableTagTR(){
    cout << "<tr>";}

  // submit タグは <td></td> の中にいれます
  void submitTag(int id){
    cout << "<td valign=center><form METHOD=\"POST\" ACTION=\"/cgi-bin/perform\">"
         << "<input type=\"HIDDEN\" name=\"id\" value=\""
 << id << "\" > "
         << "<input type=\"SUBMIT\" value=\"play\"></form></td>";}

};
```

2. fileCommon.h

ファイルのクラス CfileCommon を定義しています.

＜リスト＞

```
// c/c++ のヘッダ
#include <string.h>
#include <iostream.h>
// object store のヘッダ
#include <ostore/ostore.hh>
//
#include "base.h"

os_typespec *char_type=os_typespec::get_char();

class CfileCommon: public inHTML {
  int    id;
  char*  path;
  char*  mediaType;
  char*  format;
 public:
  static os_typespec *get_os_typespec();
  // コンストラクタ
  CfileCommon(int idNum, char* pth, char* mdType, char* frmt){
    id = idNum;

    int n = strlen(pth);
    // object store が管理する記憶領域に対し，領域を確保します.
    // 演算子 new がオーバライドされていてこれを使います
    path  = new(os_database::of(this),char_type,n+1) char[n+1];
    strcpy(path,pth);

    n = strlen(mdType);
    // object store が管理する記憶領域に対し，領域を確保します.
    mediaType = new (os_database::of(this),char_type,n+1) char[n+1];
    strcpy(mediaType,mdType);
```



```
    n = strlen(frmt);
    // object store が管理する記憶領域に対し，領域を確保します.
    format = new (os_database::of(this),char_type,n+1) char[n+1];
    strcpy(format,frmt);
  }

  // デストラクタ
  // object store が管理する記憶領域を確保していたものを，解放します.
  ~CfileCommon(){
    delete path;
    delete mediaType;
    delete format;}

  // ID 番号を表示します．表示の方法はクラス inHTML で定義されています.
  void  showID(){ hyouji(id);}
  void  showID(char* align){hyouji(id,align);}        // 位置指定用
  int   getID(){ return id;}

  // パスを表示します．表示の方法はクラス inHTML で定義されています.
  void  showPath(){hyouji(path);}
  void  showPath(char* align){hyouji(path,align);}    // 位置指定用
  char* getPath(){return path;}
//  virtual void perform();
};
```

3. audio.h

音声のクラス Caudio を定義しています.

＜リスト＞

```
/**

音声クラス Caudio を定義しています

*/

// c/c++ のヘッダ
#include <string.h>
#include <iostream.h>
// object store のヘッダ
#include <ostore/ostore.hh>
//
#include "fileCommon.h"


class Caudio: public CfileCommon{
  int    length;          // その音の長さ

  char*  item0;           // 音に対する説明（4つ）
  char*  item1;
  char*  item2;
  char*  item3;
```

```
public:
 static os_typespec *get_os_typespec();

 // コンストラクタ
 Caudio(int idNum, char* it0, char* it1, char* it2, char* it3,
int lgth, char* pth, char* frmt):
              CfileCommon(idNum,pth,"audio",frmt){

   length = lgth;

   int n = strlen(it0);
   // object store が管理する記憶領域に領域を確保します.
   item0  = new(os_database::of(this),char_type,n+1) char[n+1];
   strcpy(item0,it0);

   n =p strlen(it1);
   // object store が管理する記憶領域に領域を確保します.
   item1  = new(os_database::of(this),char_type,n+1) char[n+1];
   strcpy(item1,it1);

   n = strlen(it2);
   // object store が管理する記憶領域に領域を確保します.
   item2  = new(os_database::of(this),char_type,n+1) char[n+1];
   strcpy(item2,it2);

   n = strlen(it3);
   // object store が管理する記憶領域に領域を確保します.
   item3  = new(os_database::of(this),char_type,n+1) char[n+1];
   strcpy(item3,it3);
 }

 // デストラクタ
 // object store が管理する記憶領域に確保した領域を解放します.
 ~Caudio(){
   delete item0;
   delete item1;
   delete item2;
   delete item3;}

 // 関数名が変ですが, 音に対する説明用の項目 4 つを表示するメソッドです.
 // 表示する前にいったん object store のメモリ領域から通常のメモリ領域に
 // 表示内容をコピーしていますが, object store の管理する領域から
 // 直接読み込むと, core を吐くことがあったからです.
 void showName(){
   int n = strlen(item0);
   char *tmp0 = new char[n+1];
   strcpy(tmp0,item0);
   hyouji(tmp0);
   delete tmp0;
```



```
   n = strlen(item1);
   char *tmp1 = new char[n+1];
   strcpy(tmp1,item1);
   hyouji(tmp1);
   delete tmp1;

   n = strlen(item2);
   char *tmp2 = new char[n+1];
   strcpy(tmp2,item2);
   hyouji(tmp2);
   delete tmp2;

   n = strlen(item3);
   char *tmp3 = new char[n+1];
   strcpy(tmp3,item3);
   hyouji(tmp3);
   delete tmp3; }

 // 音の長さを表示します.
 void showLength(){
   hyouji(length);}
 void showLength(char* align){
   hyouji(length,align);}
 int getLength(){return length;}

};
```

4. aiff.h
aiff 形式の音声のクラス Caiff を定義しています. 最後のあたりで再生メソッドを定義しています. 再生メソッドでシェルスクリプトを実行しています. rsh を使って miris27 の playaifc2 を呼び出しています. 開始時刻, 終了時刻, パスはデータメンバに持っているのでそれを使います.

＜リスト＞

```
/**

Caiff は aiff 形式のファイルのクラスです

*/

// c/c++ のヘッダ
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <iostream.h>

// object store のヘッダ
#include <ostore/ostore.hh>
//
#include "audio.h"
```

```
// カプセル化したかったんですが ，結局うまくいかず大域変数になりました
class Caiff;
extern os_Set<Caiff*> *Caiff_extent;

class Caiff: public Caudio {
  int     start;                  // 開始時刻
  int     end;                    // 終了時刻

public:
  static os_typespec *get_os_typespec();
  os_Set<Caiff*> children;   // 実現していません

  // コンストラクタ
  Caiff(int idNum, char* it0, char* it1, char* it2, char* it3,int strt,
int ed, char* pth):
    Caudio(idNum,it0,it1,it2,it3,end-start,pth,"aiff"){
    start = strt;
    end   = ed;
    // このクラスのインスタンスを object store のコレクションに挿入します
    // スーパークラスでは，この命令を必要としないようです.
    Caiff_extent->insert(this);
  }
  // デストラクタ
  // object store に挿入したオブジェクトをコレクションから取り除きます.
  ~Caiff(){Caiff_extent->remove(this);}

  // 開始時刻を表示します
  void showStart(){
    hyouji(start);}
  void showStart(char* align){
    hyouji(start,align);}
  int getStart(){return start;}

  // 終了時刻を表示します
  void showEnd(){
    hyouji(end);}
  void showEnd(char* align){
    hyouji(end,align);}
  int getEnd(){
    return end;}

  // aiff 形式の再生メソッド． シェルスクリプトを実行しています.
  // リモートシェルを使って miris27 の playaifc2 を呼び出して再生しています
  void perform(){
    fork();
    execl("rsh -l guest miris27 ","/usr1/sinoue/bin/playaifc2","-s",
  start,"-e",end,getPath());}
};
```

5. property.h

クラス Cproperty を宣言しています．データベースとのやりとりをすべて，このクラスのメソッドを用い

て，行うつもりだったんですがうまくいかず結局データベースのファイル数を保持するだけのクラスになりました．データベースとのやりとりをすべてここで行う予定で，テキスト表示形式もこのクラスの基底クラスとして考えていました．データベースとのやりとりが実現できなかったおもな理由はデータメンバに os_Set<...> などのテンプレートを宣言できなかったせいです．しかし，このクラス自体を template 宣言すればできるのではないかと思います（実現はしていません）．

＜リスト＞

```
/**
データベースが保持しているファイルの数を numberOfFiles で扱います
このクラス全体がデータベースに保持されるわけではなく numberOfFiles
をデータベース中の count というルートに割り当てています.
当初はこのクラスにデータベースのルート等をカプセル化してメソッドのみで
データベースのルートを割り当てる等の処理を行なうつもりでしたが，
結局できませんでした.

*/
// C/C++ ヘッダ
#include <iostream.h>
// object store ヘッダ
#include <ostore/ostore.hh>
#include <ostore/coll.hh>

class Cproperty {
  int  *numberOfFiles;
 public:
  Cproperty(os_database *db);
  Cproperty(os_database *db, int defaultNumber);
  inline void showNumberOfFiles(void){
    cout << *numberOfFiles-1 << "  files exist.\n" ;}
  inline void incNumberOfFiles(void){
    ++*numberOfFiles;}
  inline void decNumberOfFiles(void){
    --*numberOfFiles;}
  inline int getNumberOfFiles(void){ return *numberOfFiles;}
};

 // コンストラクタ データベースから Cproperty のルートをさがして，
 // Cproperty のインスタンスを初期化します.
 Cproperty::Cproperty(os_database *db){
   // 整数型の os_typespec を取得 (static 宣言してあるみたいです)
   os_typespec *int_type = os_typespec::get_signed_int();
   // データベースから count という名前のルートを探します
   os_database_root *countRoot = db->find_root("count");
   // データベースの count というルートから値を読みます
   numberOfFiles = (int*) (countRoot->get_value(int_type));}

 // コンストラクタ Cproperty のインスタンスを作成し，データメンバの
 // データベース中のファイルの数を与えられた数に初期化します.
 // 通常は 0 を渡します．それからデータベースに Cproperty のルートを作成し，
 // それに作成した Cproperty のインスタンスを割り当てます.
```

```
Cproperty::Cproperty(os_database *db, int number){
  // 整数型の os_typespec を取得
  os_typespec *int_type = os_typespec::get_signed_int();
  // データメンバ numberOfFiles の値を領域を確保してから
  // データベースの中に入れます.
  numberOfFiles = new(db, int_type) int(number);
  // データベースにルート count を作りそこに numberOfFiles を割り当てます
  db->create_root("count")->set_value(numberOfFiles,int_type);}
```

- メイン

データベースを操作するコマンドのプログラムです. 設計が悪いので, あまり使えない思います. データベース中のエントリはすべて aiff 形式の音声であるという前提のもとで作ってしまったので, データベースのルートに Caiff という名前をつけてしまっています.

1. dbinit

データベースを初期化するコマンドです. 引数にデータベース名を指定することで, その名前を持つデータベースを初期化します.

＜リスト＞

```
/*
dbinit.cc :  データベースの初期化を行ないます.
*/

// C/C++ のヘッダ
#include <fstream.h>
#include <string.h>
#include <stdlib.h>
// object store のヘッダ
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
//
#include "aiff.h"
#include "property.h"

os_Set<Caiff*> *Caiff_extent = 0;

// 以下の関数はよくわかっていません. サンプルのものをペーストしただけです.
// A function to force vftable inclusion for collections templates

void force_vfts (void*){
  force_vfts(new os_Array<Caiff*>);
  force_vfts(new os_Bag<Caiff*>);
  force_vfts(new os_Collection<Caiff*>);
  force_vfts(new os_List<Caiff*>);
  force_vfts(new os_Set<Caiff*>);
}

int main(int argc, char *argv[])
```



```
{
  // 引数が足りないとき
  if (argc <= 1) {
    cerr << "Usage: "<< argv[0] << " filename  \n";
    return 1;}

  // ファイルの存在を調べます. ファイル存在するなら警告を出します.
  ifstream in;
  in.open(argv[1]);
  if (in) {                  // ファイルが存在したときの処理
    cerr << argv[1] <<" :  already exists \n";
    return 1;}
  in.close();

  // object store の初期化
  objectstore::initialize();
  os_collection::initialize();
  OS_ESTABLISH_FAULT_HANDLER

  // クラス Caiff の os_typespec を宣言します. object store が管理する
  // データベースとのやりとりに必ず必要です.
  os_typespec *Caiff_type  = Caiff::get_os_typespec();
  // クラス Caiff の集合の os_typespec を宣言します. これも必ず必要です
  os_typespec *Caiff_extent_type = os_Set<Caiff*>::get_os_typespec();

  // 指定されたファイル名をもつデータベースを作成します.
  os_database *db = os_database::create(argv[1]);

  // トランザクションを開始します.
  OS_BEGIN_TXN(tx1, 0, os_transaction::update)
    // Caiff のインスタンスの集合として Caiff_extent をデータベースに作成します
    Caiff_extent = &os_Set<Caiff*>::create(db);
    // Caiff_extent をデータベースのルート Caiff_extent_root に割り当てます.
    db->create_root("Caiff_extent_root")->set_value(Caiff_extent, Caiff_extent_type);

    // Cproperty のインスタンスを作成します. 存在するファイルの数は 1 にセット
    // され, データベースにルート count_root を作成し, 存在するファイルの数に
    // 割り当てます.
    Cproperty property(db, 1);
    cout << "initialized databasefile '"<< argv[1]<<"' \n";
    property.showNumberOfFiles();

  // トランザクション終了
  OS_END_TXN(tx1)

  db->close();

  OS_END_FAULT_HANDLER
  // データベースを参照するときにはファイルのパーミッションが書き込み可である
  // 必要です. ここではデータベースを初期化すると同時にパーミッションを
  // 書き込み可にします. CGI から起動される object store のプロセスの所有者と
```

```
  // データベースの所有者が常に同じであるとはかぎらないからです.
  char* script = "chmod +w ";
  strcat (script,argv[1]);
  system(script);
  return 0;
}
```

2. show

データベース中のエントリをすべて表示します. 引数にはデータベース名を指定します. データベースに対する問い合わせを行っています. ループを使って小さいものから順番に ID に対して検索を行い, そのエントリを表示します. データベースの中のエントリの順番を保つものに path というのがありますが, それは使っていません.

＜リスト＞

```
/*

show.cc : データベース中のすべてのエントリを表示

*/
// C/C++ のヘッダ
#include <fstream.h>
// object store のヘッダ
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
//
#include "aiff.h"
#include "property.h"

os_Set<Caiff*> *Caiff_extent = 0;

// 以下の関数は, サンプルからのペーストです. 良くわかっていません.
/* A function to force vftable inclusion for collections templates */
void force_vfts (void*){
  force_vfts(new os_Array<Caiff*>);
  force_vfts(new os_Bag<Caiff*>);
  force_vfts(new os_Collection<Caiff*>);
  force_vfts(new os_List<Caiff*>);
  force_vfts(new os_Set<Caiff*>);
}

int main(int argc, char *argv[])
{
  // 引数がたりないときの処理です.
  if (argc<=1){
    cerr << "Usage : "<<argv[0]<<" databasename \n";
    return 1;}

  // ファイルが存在しないときの処理です.
```



```
ifstream in;
in.open(argv[1]);
if (!in) {
  cerr << "Failed to open : "<<argv[1] <<"\n \n";
  return 1;}
in.close();

// object store の初期化
objectstore::initialize();
os_collection::initialize();
OS_ESTABLISH_FAULT_HANDLER

//データベースで使用する型の宣言
os_typespec *Caiff_extent_type = os_Set<Caiff*>::get_os_typespec();
os_typespec *Caiff_type        = new os_typespec("Caiff");

// データベースのオープン
os_database *db = os_database::open(argv[1]);

// トランザクション開始
OS_BEGIN_TXN(tx1, 0, os_transaction::update)

  // クラス Cproperty のコンストラクタにより, データベースに保持されている
  // Cproperty のデータメンバがインスタンス property に読み込まれます.
  Cproperty property(db);

  // データベース中の Caiff のルートをさがしそのポインタを Caiff_extent に
  // 代入します
  Caiff_extent = (os_Set<Caiff*>*)
    (db->find_root("Caiff_extent_root")->get_value(Caiff_extent_type));

  // データベース中のエントリ数を取り出します.
  int card = property.getNumberOfFiles();

  // 以下はデータベースへの問い合わせの処理です.
  // まず, os_coll_query 型の検索式を宣言します. 式は文字列で指定し,
  // 変数は明示的にキャストします. この場合 id が i と等しいものの問い合わせに
  // なります.
  const os_coll_query & orderedQuery =
    os_coll_query::create_pick("Caiff*","id==(int)i", db);

  Caiff *fileTmp;
  cout << "<tr>";

  // ID が小さいものから順番に取り出すためのループ
  for (int i=1; i<card; i++){
    // 問い合わせ orderedQuery の式の中の i とループ変数 i のバインド
    os_bound_query bq(orderedQuery,(os_keyword_arg("i",i)));

    // ID 番号は重複しないので, 1 項目だけ取り出す query_pick を呼び出します.
    fileTmp = Caiff_extent->query_pick(bq);
```

```
    // ID と項目 1 から項目 4 までの表示
    fileTmp->showID();
    fileTmp->showName();

    // web のページに play ボタンをつけるためのタグを生成します.
    // 個々の項目に ID 番号を書いておき，どのボタンが押されたかわかる
    // ようにします.
    fileTmp->submitTag(fileTmp->getID());
    fileTmp->kaigyou();}

  // データベース中にあるすべてのエントリ数を表示します.
  cout << "<tr><td colspan=6 align=center>";

  property.showNumberOfFiles();
  cout << "</td></tr>";

 // トランザクション終了
 OS_END_TXN(tx1)

 // データベースを閉じる
 db->close();

 OS_END_FAULT_HANDLER
 return 0;
}
```

3. id

引数でデータベース名と，2 つの ID 番号を指定します．指定された ID 番号を検索の開始と終了とし，その範囲にある ID を持つエントリを表示します．このプログラムも show と同じ方法で検索を行っています．

＜リスト＞

```
/*

id.cc 引数で指定された範囲に該当する id を持つエントリを表示する

*/
// C/C++ のヘッダ
#include <fstream.h>
#include <stdlib.h>
// object store のヘッダ （順番はこの順番にインクルードするそうです）
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
//
#include "aiff.h"
#include "property.h"

os_Set<Caiff*> *Caiff_extent = 0;
```



```
// 以下の関数は，サンプルからのペーストです．良くわかっていません．
/* A function to force vftable inclusion for collections templates */
void force_vfts (void*){
  force_vfts(new os_Array<Caiff*>);
  force_vfts(new os_Bag<Caiff*>);
  force_vfts(new os_Collection<Caiff*>);
  force_vfts(new os_List<Caiff*>);
  force_vfts(new os_Set<Caiff*>);
}

int main(int argc, char *argv[])
{
  // ファイルが存在しないときの処理
  ifstream in;
  in.open(argv[1]);
  if (!in) {
    cerr << "Failed to open : "<<argv[1] <<"\n \n";
    return 1;}
  in.close();

  // object store の初期化
  objectstore::initialize();
  os_collection::initialize();
  OS_ESTABLISH_FAULT_HANDLER

  // データベースで使用する型の宣言
  os_typespec *Caiff_extent_type = os_Set<Caiff*>::get_os_typespec();

  // こういうふうに宣言することもできるそうです.
  os_typespec *Caiff_type  = new os_typespec("Caiff");

  // データベースのオープン
  os_database *db = os_database::open(argv[1]);

  // トランザクション開始
  OS_BEGIN_TXN(tx1, 0, os_transaction::update)

    // クラス Cproperty のコンストラクタにより，データベースに保持されている
    // Cproperty のデータメンバがインスタンス property に読み込まれます.
    Cproperty property(db);

    // データベース中の Caiff のルートをさがしそのポインタを Caiff_extent に
    // 代入します
    Caiff_extent = (os_Set<Caiff*>*)
      (db->find_root("Caiff_extent_root")->get_value(Caiff_extent_type));

    // 検索する ID の範囲です． 引数で与えられます.
    int minimum, maximum;
    minimum = atoi(argv[2]);
    maximum = atoi(argv[3]);
```

```
  // 以下はデータベースへの問い合わせの処理です．
  // まず，os_coll_query 型の検索式を宣言します．式は文字列で指定し，
  // 変数は明示的にキャストします．
  const os_coll_query & orderedQuery =
    os_coll_query::create_pick("Caiff*","id==(int)i", db);

  // ID の小さいものから順に検索していきます．
  // object store の「パス」という機能を利用すればもっと簡単に実現できる
  // ようです．
  Caiff *fileTmp;
  for (int i=minimum; i<=maximum; i++){
    // 変数のバインド． 上で指定した検索条件で使用した変数 i をループ変数
    // i にバインドします．変数のほとんどはバインドしないと実行時エラー
    // を起こします．
    os_bound_query bq(orderedQuery,(os_keyword_arg("i",i)));

    // query_pick は検索条件にあうものを 1 つだけ取り出します．
    // この場合，重複した ID 番号を持つエントリはないので，これを使います．
    fileTmp = Caiff_extent->query_pick(bq);

    // 1 行の始まりを指定します．
    cout << "<tr>";

    fileTmp->showID();
    fileTmp->showName();

    // web のページに play ボタンをつけるためのタグを生成します．
    // 個々の項目に ID 番号を書いておき，どのボタンが押されたかわかる
    // ようにします．
    fileTmp->submitTag(fileTmp->getID());

    // 1 行の終りを指定します．
    cout << "</tr>" << endl;}

  // トランザクション終了
  OS_END_TXN(tx1)

  // データベースを閉じる
  db->close();

  OS_END_FAULT_HANDLER
  return 0;
}
```

4. item

引数でデータベース名，検索文字列，検索対象項目，検索開始 ID，検索終了 ID を指定し，条件にあうエントリを出力します．検索方法は，項目 1 から項目 4 までのうち指定された検索の対象項目について検索を行い，最終的に指定された検索対象項目に検索文字列を含むすべてのエントリを出力します．

＜リスト＞//

```
/*
```



```
ID 番号を入れて該当する項目を出力します．
引数は データベース名 検索文字列 検索対象項目 1 (true または false)
  ... 検索対象項目 4 [検索開始 ID] [検索終了 ID]
開始，終了を指定しない時には，データベース中のすべてが対象になります．

*/
// C/C++ のヘッダ
#include <fstream.h>
#include <stdlib.h>
// object store のヘッダ
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
//
#include "aiff.h"
#include "property.h"

//  引数が指す内容と順番の対応．左が内容で右が引数の順番です．
#define  DB_NAME   1
#define  STRING    2
#define  ITEM1     3
#define  ITEM2     ITEM1+1
#define  ITEM3     ITEM1+2
#define  ITEM4     ITEM1+3
#define  BEGIN     ITEM1+4
#define  END       ITEM1+5

os_Set<Caiff*> *Caiff_extent = 0;

// 以下の関数は，サンプルからのペーストです．良くわかっていません．
/* A function to force vftable inclusion for collections templates */
void force_vfts (void*){
  force_vfts(new os_Array<Caiff*>);
  force_vfts(new os_Bag<Caiff*>);
  force_vfts(new os_Collection<Caiff*>);
  force_vfts(new os_List<Caiff*>);
  force_vfts(new os_Set<Caiff*>);
}

int main(int argc, char *argv[])
{

  // 指定されているときには，検索開始 ID と検索終了 ID を代入する．
  int begin;
  int end;
  if (argc==END+1){
    begin = atoi(argv[BEGIN]);
    end   = atoi(argv[END]);}

  // ファイルが存在しないときの処理です．
  ifstream in;
```

```
in.open(argv[DB_NAME]);
if (!in) {
  cerr << "Failed to open : "<<argv[DB_NAME] <<"\n \n";
  return 1;}
in.close();

// object store の初期化の処理です.
objectstore::initialize();
os_collection::initialize();
OS_ESTABLISH_FAULT_HANDLER

// データベースで使用する型の宣言
os_typespec *Caiff_extent_type = os_Set<Caiff*>::get_os_typespec();
os_typespec *Caiff_type  = new os_typespec("Caiff");

// データベースのオープン
os_database *db = os_database::open(argv[DB_NAME]);

// トランザクション開始
OS_BEGIN_TXN(tx1, 0, os_transaction::update)

  // クラス Cproperty のコンストラクタにより，データベースに保持されている
  // Cproperty のデータメンバがインスタンス property に読み込まれます.
  Cproperty property(db);

  // Caiff のルートをみつけ，それに対するポインタを Caiff_extent に
  // 代入します
  Caiff_extent = (os_Set<Caiff*>*)
    (db->find_root("Caiff_extent_root")->get_value(Caiff_extent_type));

  // result を Caiff の集合として宣言
  // result には全項目において検索文字列にあてはまるものをいれます
  // 下で宣言する files を result の中に加えていく
  os_Collection<Caiff*>&result = os_Collection<Caiff*>::create(db);

  // files を Caiff の集合として宣言．検索結果をいれます.
  // files には 1 つの項目の中で検索文字列にあてはまるものだけを入れる
  os_Collection<Caiff*>&files = os_Collection<Caiff*>::create(db);

  // まず containsQuery を宣言する．Caiff で strstr(...) に該当するものを
  // さがすことを宣言します．変数名，関数名は明示的にキャストします.
  // 項目 1 から項目 4 までのそれぞれに検索をして，結果を files に加えます.
  // files はコレクションなので重複する項目を挿入しても files の内容
  // はかわりません．項目 1 から項目 4 までのそれぞれの検索結果を
  // result に加え，結果を得ます.

  // 項目 1 を検索するためのクエリーを作ります
  const os_coll_query & containsQuery0 =
    os_coll_query::create("Caiff*",
   "(char*)strstr(item0,(char*)words)!=(char*)NULL", db);
```



```
  // 項目 2 を検索するためのクエリーを作ります
  const os_coll_query & containsQuery1 =
    os_coll_query::create("Caiff*",
   "(char*)strstr(item1,(char*)words)!=(char*)NULL", db);

  // 項目 3 を検索するためのクエリーを作ります
  const os_coll_query & containsQuery2 =
    os_coll_query::create("Caiff*",
   "(char*)strstr(item2,(char*)words)!=(char*)NULL", db);

  // 項目 4 を検索するためのクエリーを作ります
  const os_coll_query & containsQuery3 =
    os_coll_query::create("Caiff*",
          "(char*)strstr(item3,(char*)words)!=(char*)NULL", db);


  // 次に containsQuery の中で使っていた変数 words を検索文字列を代入した
  // 変数 string にバインドし，os_bound_query 型の bq を宣言します.
  // 以上で検索の準備は完了です.

  // 項目 1 を検索するためのクエリーのバインド
  os_bound_query bq0(containsQuery0,(
os_keyword_arg("words",argv[STRING]),
                os_keyword_arg("strstr",strstr),
                os_keyword_arg("NULL",NULL)));

  // 項目 2 を検索するためのクエリーのバインド
  os_bound_query bq1(containsQuery1,(
os_keyword_arg("words",argv[STRING]),
                os_keyword_arg("strstr",strstr),
                os_keyword_arg("NULL",NULL)));

  // 項目 3 を検索するためのクエリーのバインド
  os_bound_query bq2(containsQuery2,(
os_keyword_arg("words",argv[STRING]),
                os_keyword_arg("strstr",strstr),
                os_keyword_arg("NULL",NULL)));

  // 項目 4 を検索するためのクエリーのバインド
  os_bound_query bq3(containsQuery3,(
os_keyword_arg("words",argv[STRING]),
                os_keyword_arg("strstr",strstr),
                os_keyword_arg("NULL",NULL)));


  // 実際に bq を使って項目 1 に問い合わせをします
  // もし引数の 1 文字目が "t"なら検索を実行します
  if ( argv[ITEM1][0]=='t' ){
    files = Caiff_extent->query(bq0);
    result|=files;}
```

```
// 実際に bq を使って項目 2 に問い合わせをします
// もし引数の 1 文字目が "t"なら検索を実行します
if ( argv[ITEM2][0]=='t' ){
  files = Caiff_extent->query(bq1);
  result|=files;}

// 実際に bq を使って項目 3 に問い合わせをします
// もし引数の 1 文字目が "t"なら検索を実行します
if ( argv[ITEM3][0]=='t' ){
  files = Caiff_extent->query(bq2);
  result|=files;}

// 実際に bq を使って項目 4 に問い合わせをします
// もし引数の 1 文字目が "t"なら検索を実行します
if ( argv[ITEM4][0]=='t' ){
  files = Caiff_extent->query(bq3);
  result|=files;}

// result の中の要素すべてを取り出すためのループの準備
int   numberOfFiles    = property.getNumberOfFiles();
// 検索に引っかかった項目の数
int   numberOfDetected = result.cardinality();
Caiff *fileTmp;

// result の中の要素すべてを順番にみていくためのクエリーとバインド
/*** os_Collection<E>. のメソッドの中に順番を保つ「パス」という機能が ***/
/*** ありますが使ってません.                                          ***/

// ID を指定してその ID をもつエントリを取り出すためのクエリー
const os_coll_query & orderedQuery =
  os_coll_query::create_pick("Caiff*","id==(int)i", db);

// 指定した ID がコレクションの中に存在するかを調べるためのクエリー
const os_coll_query & existanceQuery =
  os_coll_query::create_exists("Caiff*","id==(int)i", db);

// 引数が ID 番号が指定されてない場合の検索開始 ID と検索終了 ID
if (argc!=END+1){
  begin = 1;
  end   = numberOfFiles;}


// 検索されたエントリ数の初期化
int detected =0;
cout << "<tr>";

// result の中の要素すべてを順番に取り出すためのループ
for (int i=begin; i<=end; i++){
  // ID を指定してその ID をもつエントリを取り出すためのバインド
  os_bound_query bqExists(existanceQuery,(os_keyword_arg("i",i)));
```



```
    if (result.exists(bqExists)!=0){  // ID が i のエントリの存在のチェック
// 存在するときにはその ID をもつエントリを取り出すためのバインド
// を行ないます.
os_bound_query bqPick(orderedQuery,(os_keyword_arg("i",i)));

// 実際に検索を行ないます.
fileTmp = result.query_pick(bqPick);

// ID と項目 1 から項目 4 を出力します.
fileTmp->showID();
fileTmp->showName();

// web のページに play ボタンをつけるためのタグを生成します.
// 個々の項目に ID 番号を書いておき, どのボタンが押されたかわかる
// ようにします.
fileTmp->submitTag(fileTmp->getID());

fileTmp->kaigyou();
        detected++;}
}

    // 問い合わせに引っかかったエントリの数を表示
    cout << "<tr><td align=center colspan=4>" << detected
         << " files detected  </td></tr>\n";
  OS_END_TXN(tx1)

  db->close();

  OS_END_FAULT_HANDLER
  return 0;
}
```

5. perform

引数で指定されたデータベースの中の指定された ID を持つエントリの再生メソッドを呼び出します. 検索方法は id と性質同じもので, 表示するかわりに再生メソッドを呼び出します.

＜リスト＞//

```
/*

perform.cc 入力された数字に該当する id を持つエントリの再生メソッドを呼び出す.

*/
// C/C++ のヘッダ
#include <fstream.h>
#include <stdlib.h>
// object store のヘッダ
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
//
#include "aiff.h"
#include "property.h"
```

```
os_Set<Caiff*> *Caiff_extent = 0;

/ サンプルにあったもののペーストです.
/* A function to force vftable inclusion for collections templates */
void force_vfts (void*){
  force_vfts(new os_Array<Caiff*>);
  force_vfts(new os_Bag<Caiff*>);
  force_vfts(new os_Collection<Caiff*>);
  force_vfts(new os_List<Caiff*>);
  force_vfts(new os_Set<Caiff*>);
}

int main(int argc, char *argv[])
{
  // ファイルが存在しないときの処理です.
  ifstream in;
  in.open(argv[1]);
  if (!in) {
    cerr << "Failed to open : "<<argv[1] <<"\n \n";
    return 1;}
  in.close();

  // object store の初期化
  objectstore::initialize();
  os_collection::initialize();
  OS_ESTABLISH_FAULT_HANDLER

  // データベースで使用する型の宣言
  os_typespec *Caiff_extent_type = os_Set<Caiff*>::get_os_typespec();
  os_typespec *Caiff_type  = new os_typespec("Caiff");

  // データベースをオープン
  os_database *db = os_database::open(argv[1]);

  // トランザクション開始
  OS_BEGIN_TXN(tx1, 0, os_transaction::update)

    // クラス Cproperty のコンストラクタにより，データベースに保持されている
    // Cproperty のデータメンバがインスタンス property に読み込まれます.
    Cproperty property(db);

    // データベース中の Caiff のルートをさがしそのポインタを Caiff_extent に
    // 代入します.
    Caiff_extent = (os_Set<Caiff*>*)
      (db->find_root("Caiff_extent_root")->get_value(Caiff_extent_type));

    // 引数で指定された ID を代入します.
    int target = atoi(argv[2]);

    // 以下はデータベースへの問い合わせの処理です.
```



```
    // まず，os_coll_query 型の検索式を宣言します．式は文字列で指定し，
    // 変数は明示的にキャストします.
    const os_coll_query & orderedQuery =
      os_coll_query::create_pick("Caiff*","id==(int)target", db);

    Caiff *fileTmp;

    // 変数のバインド．　上で指定した検索条件で使用した変数 i をループ変数
    // i にバインドします．変数の殆どんどはバインドしないと実行時エラー
    // を起こします.
    os_bound_query bq(orderedQuery,(os_keyword_arg("target",target)));

    // query_pick は検索条件にありものを 1 つだけ取り出します.
    // この場合，重複した ID 番号を持つエントリはないので，これを使います.
    fileTmp = Caiff_extent->query_pick(bq);

    // 再生メソッドを呼び出します.
    fileTmp->perform();

  // トランザクション終了
  OS_END_TXN(tx1)

  // データベースを閉じる
  db->close();

  OS_END_FAULT_HANDLER
  return 0;
}
```

説明に不十分な部分がありましたら、shuuji-m@is.aist-nara.ac.jp までメールをください.