ホロン株式会社

HiBase API for Java v1.0
プログラミングリファレンス

本マニュアル、およびマニュアルに記載されているソフトウェア（コンピュータプログラム）は、ホロン株式会社にすべての権利が帰属します。

ソフトウェアの正常な使用またはバックアップコピーを作成する場合を除き、ホロン株式会社の書面による同意なしには、本マニュアルやプログラムの一部または全部を問わず、複製を禁じています。
本ソフトウェアの仕様、およびマニュアルに記載されている事項は、将来予告なしに変更することがあります。
本マニュアルには、正確な情報を記載するように努めましたが、誤植や制作上の誤記がないことを保証するものではありません。
また、本ソフトウェアおよびマニュアルを運用した結果の影響については、一切責任を負いかねますのでご了承ください。

Appleの名称、ロゴマークは、米国その他の国で登録された米国アップルコンピュータ社の登録商標です。
その他、本マニュアルに記載されたソフトウェア、ハードウェア製品の名称およびロゴマークは、すべて開発および著作・販売会社等、関係各社の商標または登録商標であることを明記し、本文中での表記を省略させていただきます。

本マニュアルに記載された人物、団体名は、全て架空であり、実在いたしません。
類似の人物、団体名が存在する場合は、全くの偶然であり、本マニュアルには一切関係がありません。
また、本マニュアルに記載された他社商品名は、参考を目的としたものであり、それら製品の使用を強制または推奨するものではありません。

目次

はじめに

この文章は、Java言語からRDBMS（リレーショナルデータベース管理システム） - HiBaseを利用するためのAPI（アプリケーションプログラムインタフェース）について説明する。

HiBase API for Java は純粋にJava 言語で記述されており、TCP/IP 通信をもちいて HBServer（HiBaseサーバ）に接続し、データベース管理やトランザクション処理をおこなうことが可能だ。HiBase API for Java はJDBCや ODBCを利用せずに、HiBaseサーバ（HBServer）のサービスをダイレクトに呼び出す。そのため HiBaseの高速性と柔軟性を生かしたまま、Java 言語の利点をそのまま全て生かすことができる。

HiBase API for Java はJDK 1.0 以降を対象として開発されている。そのため、Java Application からの利用はもちろんのこと、JDK 1.1 に完全に対応していない多くの Webブラウザ上でも、Java Appletからのデータベース接続が行える。

HiBase API はC/C++ で記述されており、そのままの形での Java言語への移植は困難、かつ無理がある。HiBase API for Javaでは。HiBase API のフレームワークはそのままに、Java言語からの利用を考えた仕様になっている。例えば、HiBase API の特徴の一つである、メモリ上へのダイレクトI/O を行わずに、Java言語の String クラスを用いたI/O を行うなどの変更がなされている。また、これらのクラスは Javaのパッケージになっており、jp.co.hln.hibase の元に構築されている。HiBase API for Java を利用する場合は、適切な import文が必要だ。

HiBaseの概念、用語など、詳しくは「HiBaseプログラミングガイド」または、「HiBase プログラミングリファレンス」を参照。

１．HiBase への接続　- HClient

HiBaseに接続するには、HClientオブジェクトが必要だ。HClientクラスは、HBServerが稼働しているホストマシンのホスト名（もしくは IP アドレス）と、データベースサービスのポート番号（デフォルトは3330）を引数として指定し、生成される。

HClientオブジェクトの生成後、open()メソッドを実行することにより、HBServerのデータベースサービスへの接続がおこなわれる。終了時には、close()メソッドを実行することにより、データベースサービスとの接続を切断する。

１．１　コンストラクタ

```
public HClient( String hostName, int portNo );
```

解説

HClientクラスのコンストラクタ。hostNameに接続対象のホスト名（もしくはIPアドレス）を指定。portNoにはデータベースサービスのポート番号を指定。デフォルトのポート番号は3330としている。

ホスト名やポート番号は、サーバの稼働しているホストマシンやHBServerの設定、ネットワークの設定に依存する。

１．２　接続、切断

```
public void open() throws IOException, UnknownHostException;
```

解説

データベースサービスへの接続をおこなう。例外が発生せず、error()メソッドが0(HError.ecNormal)を返すときに、データベースサービスへの接続が成功している。

```
public void close() throws IOException;
```

解説

データベースサービスへの接続を切断する。例外が発生せず、error()メソッドが0(HError.ecNormal)を返すときに、サービスの切断が成功している。

```
public boolean isOpened()
```

解説

データベースサービスに接続しているかどうかを調る。戻り値がtrueの時に接続済み、falseの時

に未接続を示している。

１．３　アクセッサ

```
public int error();
```

解説

エラーコードを取得する。戻り値が０(HError.ecNormal)の時、エラーは発生していない。

エラーコードの詳細は。HError.javaを参照。

```
public String getHostName();
```

解説

ホスト名を取得する。コンストラクタに指定したホスト名を返す。

```
public int getPortNumber();
```

解説

ポート番号を取得する。コンストラクタに指定したポート番号を返す。

２．HiBase にアクセス　- HDBHandle

HDBHandle クラスはデータベース中の定義情報やデータの処理に関連する機能を提供する。HDBHandle クラスの提供する機能は大きく分けて、DD/D 関連と DML関連に分類できる。

DD/D 関連機能は、HiBaseのデータベース構成要素である、データベースファイル、アイテム、キーの定義情報の登録／削除／変更／読みだしをおこなう。またさらに、データベースファイル、キーの生成／削除をおこなう。

DML関連機能は、データベースに格納されているデータの追加／削除／更新／検索／読みだしなどのトランザクション処理をおこなう。

HiBaseサーバとの接続に成功した HClientオブジェクトを引数に指定して、HDBHandleオブジェクトを生成する。その後、HDBHandle.open()メソッドを実行して、データベースをオープンする。データベースに対する処理が終了した時点で、HDBHandle.close()メソッドを実行して、データベースをクローズしなければならない。

２．１　コンストラクタ

```
public HDBHandle( HClient inSocket );
public HDBHandle( HClient inSocket, short inDBNbr );
```

解説

HDBHandle クラスのコンストラクタ。HClientオブジェクトを引数として渡すことにより、HBServerとのデータベースサービスを行う。２種類のコンストラクタが用意されている。最初の、引数が HClientのみのコンストラクタは、HBServerのデフォルトデータベースを対象とする。２番目のコンストラクタは、データベース番号を指定し、特定のデータベースを操作の対象とすることが出来る。HiBase v4では、デフォルトデータベースへの処理が基本となる。

２．２　データベース処理

```
public boolean isDBExist() throws IOException;
```

解説

データベースが既に存在しているかどうかを確認する。戻り値が、true の時、データベースは存在しており、false の時、存在していないことを示す。

```
public HDBInfo dbGetInfo() throws IOException;
```

解説

データベース情報を取得する。HDBInfo オブジェクトには、データベースの名前、データベースのコメント、データベースが生成されているかどうかのフラグが含まれる。

HDBInfo クラスは以下のようになっている。

```
public class HDBInfo {
  public String  dbName;                  // database name
  public String  dbDesc;                  // database desc
  public boolean dbCreated;               // database was created

  public HDBInfo();
}
```

dbName、dbDesc、dbCreatedはpublic 宣言されているので、これらを設定／参照することが出来る。

```
  public void dbSetInfo( HDBInfo inInfo ) throws IOException;
```

解説

データベース情報を設定する。HDBInfo オブジェクトを引数として渡し、データベースの名前、データベースのコメントを指定することが出来る。

```
  public void dbClearInfo() throws IOException;
```

解説

データベース情報を消去する。

```
public void dbCreate() throws IOException;
```

解説

データベース情報に基づき、データベースを生成する。このメソッドの実行に先立ち、dbSetInfo ()メソッドでデータベース情報が設定されてなければならない。

```
public void dbDelete() throws IOException;
```

解説

データベースを消去する。このメソッドの実行により、データベースファイル、アイテム、キーも消去される。また同様に、データベースファイル定義、アイテム定義、キー定義も消去される。

このメソッドを実行しても、データベース定義は消去されない。データベース定義を消去するには、このメソッドの実行後、dbClearInfo()を実行する必要がある。dbDelete()の実行直後に dbCreate()を実行することにより、データベースを初期化することが出来る。

```
public void dbSetActive( short inDBNbr ) throws IOException;
```

解説

inDBNbr で示す番号のデータベースをアクティブデータベースにする。

```
public short dbGetActive() throws IOException;
```

解説

アクティブデータベースのデータベース番号を取得する。

```
public boolean isOpened();
```

解説

データベースがオープンしているかどうかを問い合わせる。戻り値がtrue の時、データベースは開いており、false の時、データベースは開かれていない。データベースのオープン／クローズは、open()／close()メソッドにより行う。

```
public void open() throws IOException;
```

解説

データベースを開く。データベースに対する全ての処理の前に、このメソッドを実行する必要がある。これにより、データベースとの接続が行なわれる。

```
public void close() throws IOException;
```

解説

データベースを閉じる。データベースに対する処理が終了した後に、このメソッドを実行する必要がある。close()の実行で、データベースとの接続が切断され、これ以降のデータベースに対する処理は全て無効になる。

```
public void trCommit() throws IOException;
```

解説

データベースに対するトランザクション処理の正当性をデータベースに委任する。一連のトランザクション処理で、レコードの追加、更新、削除などを行った後、その一連の処理を確定するために、trCommit()を実行する必要がある。

一連のトランザクション処理中に、なんらかのエラーが発生した場合、処理の先頭に戻すためには、trCancel()を実行する。

```
public void trCancel() throws IOException;
```

解説

データベースに対して行った、一連のトランザクション処理を取り消す。

２．３　データベースの定義と生成

２．３．１　データベースファイル定義情報の操作

```
  public HFileDefine ddGetFile( int inFileNbr ) throws IOException;
```

解説

HiBaseから、inFileNbrで示すデータベースファイル定義情報を読みだす。戻り値の、HFileDefineはデータベースファイル定義情報を格納しているオブジェクトである。

HFileDefileクラスは、以下のようになっている。

```
public class HFileDefine {
  public int     fileNbr;                // file number
  public String  fileName;               // file name
  public String  fileDesc;               // file desc
  public short   fileIBSize;             // index block size
  public short   fileDBSize;             // data  block size
  public boolean   fileCreated;          // file was created

  public HFileDefine();
}
```

fileNbr、fileName、fileDesc、fileIBSize、fileDBSize、fileCreatedはpublicと宣言されているので、これらを設定／参照することが出来る。

※詳しくは、「HiBaseプログラミングリファレンス」-「DD/D関連の定義情報の構造」を参照

```
  public void ddAddFile( HFileDefine inDef ) throws IOException;
```

解説

HiBaseに、inDefで示すデータベースファイル定義情報を登録する。実行に先立ち、HFileDefineオブジェクトを生成し、必要なメンバ変数を設定しておく必要がある。

```
public void ddDelFile( int inFileNbr ) throws IOException;
```

解説

HiBaseから、inFileNbr で示すデータベースファイルの定義情報を削除する。

このとき、データベースファイル定義情報を削除するだけでなく、ファイル定義に含まれるすべてのアイテム定義情報、キー定義情報も削除する。

```
public void ddUpdFile( int inFileNbr, HFileDefine inDef )
throws IOException;
```

解説

HiBaseの inFileNbrで示すデータベースファイル定義情報を、inDefで示す定義情報に更新する。

```
public int[] ddListFile() throws IOException;
```

解説

HiBaseに登録されているすべてのデータベースファイル番号をとりだす。

戻り値は、int型の配列で、データベースファイルのファイル番号を格納した配列である。データベースファイルの個数は、その配列の大きさで示される。

２．３．２　アイテム定義情報の操作

```
  public HItemDefine ddGetItem( int inFileNbr, short inItemID )
  throws IOException;
```

解説

HiBaseのinFileNbr で示されるデータベースファイルに含まれる、inItemIDで示されるアイテム定義情報をとりだす。戻り値は、アイテム定義情報を格納した HItemDefineオブジェクトである。

HItemDefine クラスは、以下のようになっている。

```
public class HItemDefine {
  public short   iid;                    // item id
  public short   itmType;                // data type
  public int     itmLength;              // data length

  public String  itmName;                // item name
  public String  itmDesc;                // item desc
  public boolean   itmCreated;           // item was created (not used)

  public HItemDefine();
}
```

iid、itmType、itmLength、itmName、itmDesc、itmCreatedはpublic と宣言されているので、これらを設定／参照することが出来る。

※詳しくは、「HiBaseプログラミングリファレンス」-「DD/D 関連の定義情報の構造」を参照

```
  public void ddAddItem( int inFileNbr, HItemDefine inDef )
  throws IOException;
```

解説

HiBaseの inFileNbrで示されるデータベースファイルに、inDefで示すアイテム定義情報を登録する。実行に先立ち、HItemDefine オブジェクトを生成し、必要なメンバ変数を設定しておかねばならない。

```
public void ddDelItem( int inFileNbr, short inItemID )
throws IOException;
```

解説

HiBaseのinFileNbr で示されるデータベースファイルから、inItemID で示されるアイテム定義情報を削除する。

```
public void ddUpdItem( int inFileNbr, short inItemID,
        HItemDefine inDef ) throws IOException;
```

解説

HiBaseのinFileNbr で示されるデータベースファイルから、inItemID で示されるアイテム定義情報を、inDefで示す定義情報に更新する。

```
public short[] ddListItem( int inFileNbr ) throws IOException;
```

解説

HiBaseのinFileNbr で示されるデータベースファイルから、登録されているすべてのアイテム番号をとりだす。

戻り値は、short 型の配列で、アイテム番号を格納した配列である。アイテムの個数は、その配列の大きさで示される。

２．３．３　キー定義情報の操作

```
  public HKeyDefine ddGetKey( int inFileNbr, short inKeyID )
  throws IOException;
```

解説

HiBaseのinFileNbrで示されるデータベースファイルから、inKeyIDで示されるキー定義情報をとりだす。

HKeyDefineクラスは、以下のようになっている。

```
public class HKeyDefine {
  public short   kid;                    // key id
  public short   keyType;                // key type
  public short   nItm;                   // nbr of items to bind
  public HItemBind[]    itmBind;         // binding rule

  public String  keyName;                // key name
  public String  keyDesc;                // key desc
  public boolean   keyCreated;           // key was created

  public HKeyDefine();
}
public class HItemBind {
  public short   iid;                    // item id
  public short   iStart;                 // from (1 start)
  public short   iEnd;                   // to   (1 start)
  public boolean[] iOpt;                 // reserved for sort

  public HItemBind();
}
```

kid、keyType、nItm、itmBind、keyName、keyDesc、keyCreatedはpublicと宣言されているので、これらを設定／参照することが出来る。また、HItemBindクラスのiid、iStart、iEndも同様。

※詳しくは、「HiBaseプログラミングリファレンス」-「DD/D関連の定義情報の構造」を参照

```
public void ddAddKey( int inFileNbr, HKeyDefine inDef )
throws IOException;
```

解説

HiBaseの inFileNbrで示されるデータベースファイルに、inDefで示されるキー定義情報を追加する。

```
public void ddDelKey( int inFileNbr, short inKeyID ) throws IOException;
```

解説

HiBaseのinFileNbr で示されるデータベースファイルから、inKeyIDで示されるキー定義情報を削除する。

```
public void ddUpdKey( int inFileNbr, short inKeyID, HKeyDefine inDef )
throws IOException;
```

解説

HiBaseのinFileNbr で示されるデータベースファイルの、inKeyIDで示されるキーの定義情報をinDefで示される定義情報に更新する。

```
public short[] ddListKey( int inFileNbr ) throws IOException;
```

解説

HiBaseのinFileNbr で示されるデータベースファイルから、登録されているすべてのキー番号をとりだす。

戻り値は、short 型の配列で、キー番号を格納した配列である。キーの個数は、その配列の大きさで示される。

```
public void ddAddKeyFromItem( int inFileNbr,
        short inKeyID, short inItemID ) throws IOException;
```

解説

HiBaseのinFileNbr で示されるデータベースファイルに、すでに登録されているアイテム情報から、キー情報を生成し、登録する。

inKeyIDで登録するキー番号を指定し、inItemID で元になるアイテム番号を指定する。

２．３．４　データベースファイルの生成／削除

```
public void dbFileCreate( int inFileNbr ) throws IOException;
```

解説

HiBaseデータベース内に、inFileNbrで示すデータベースファイルを生成する。

このメソッドの実行に先立ち、データベースファイル定義情報が登録されてなければならない。また、このデータベースファイルに関してのアイテム定義情報、キー定義情報が登録されているならば、それらも生成される。

```
public void dbFileDelete( int inFileNbr ) throws IOException;
```

解説

HiBaseデータベース内の、inFileNbrで示すデータベースファイルを削除する。

このメソッドの実行により、データベースファイル定義情報、アイテム定義情報、キー定義情報の削除は行なわれない。

２．３．５　定義情報の操作関数

```
public int ddFileName2ID( String inName ) throws IOException;
```

解説

データベースファイル名から、データベースファイル番号をとりだす。

```
public String ddFileID2Name( int inFileNbr ) throws IOException;
```

解説

データベースファイル番号から、データベースファイル名をとりだす。

```
public short ddItemName2ID( int inFileNbr, String inName )
throws IOException;
```

解説

inFileNbr で示されるデータベースファイル内の、inName で示されるアイテム名から、アイテム番号をとりだす。

```
public String ddItemID2Name( int inFileNbr, short inItemID )
throws IOException;
```

解説

inFileNbr で示されるデータベースファイル内の、inItemID で示されるアイテム番号から、アイテム名をとりだす。

```
public short ddKeyName2ID( int inFileNbr, String inName )
throws IOException;
```

解説

inFileNbr で示されるデータベースファイル内の、inName で示されるキー名から、キー番号をとりだす。

```
public String ddKeyID2Name( int inFileNbr, short inKeyID )
throws IOException;
```

解説

inFileNbr で示されるデータベースファイル内の、inKeyIDで示されるキー番号から、キー名をとりだす。

２．３．６　キーの生成／削除

HiBaseでは、データベースファイルの生成と／削除の際に、自動的にキーも生成／削除される。

しかし、以下のメソッドを用いて、キーの生成／削除を行うことも出来る。

```
public void createAllKey( int inFileNbr ) throws IOException;
```

解説

inFileNbr で示されるデータベースファイル内のすべてのキーを生成する。

このメソッドの実行に先立ち、キー定義情報が登録されている必要がある。

```
public void deleteAllKey( int inFileNbr ) throws IOException;
```

解説

inFileNbr で示されるデータベースファイル内のすべてのキーを削除する。

このメソッドの実行により、キー定義情報は削除されない。

```
public void createKey( int inFileNbr, short inKeyID )
throws IOException;
```

解説

inFileNbr で示されるデータベースファイル内の、inKeyIDで示されるキーを生成する。

このメソッドの実行に先立ち、キー定義情報が登録されている必要がある。

```
public void deleteKey( int inFileNbr, short inKeyID )
throws IOException;
```

解説

inFileNbr で示されるデータベースファイル内の、inKeyIDで示されるキーを削除する。

このメソッドの実行により、キー定義情報は削除されない。

２．４　トランザクション処理

HiBaseに対するデータの追加／削除／更新／読みだしはレコード単位で行われる。レコードは、HRecordクラスで表現されており、クライアント側のユーザは、HRecordクラスを継承したHXRecord クラスを利用する。ここでは、レコードをどのように I/O するかを説明する。

```
public int maxRNbr( int inFileNbr ) throws IOException;
```

解説

HiBaseデータベースファイルの、レコード発番数を得る。

HiBaseデータベースファイルのレコードは、データベースファイル内でユニークな番号（レコード番号）で管理されている。このメソッドは、レコード番号の現在発行した最大の番号を返す。

```
public int fileSize( int inFileNbr ) throws IOException;
```

解説

HiBaseデータベースファイル内の、レコードの総数を返す。

```
public void getRecord( int inFileNbr, int inRNbr, HRecord outRecord )
throws IOException;
```

解説

inFlleNbr で示されるデータベースファイルから、inRNbr で示されるレコード番号のレコードを読みだして、outRecordが示す HRecordオブジェクトに返す。

通常（クライアント側のユーザは）、outRecordにはHXRecord オブジェクトを渡す。

```
public int insRecord( int inFileNbr, HRecord inRecord )
throws IOException;
```

解説

inFileNbr で示されるデータベースファイルに、inRecordが示すHRecordオブジェクトを追加する。追加されたレコードのレコード番号が、戻り値として返される。

通常（クライアント側のユーザは）、inRecordにはHXRecordオブジェクトを渡す。

```
public void updRecord( int inFileNbr, int inRNbr, HRecord inRecord )
throws IOException;
```

解説

inFileNbr で示されるデータベースファイル中の inRNbrで示されるレコード番号のレコードを、inRecordが示す HRecordオブジェクトに更新する。

通常（クライアント側のユーザは）、inRecordにはHXRecordオブジェクトを渡す。

```
public void delRecord( int inFileNbr, int inRNbr ) throws IOException;
```

解説

inFileNbr で示されるデータベースファイル中の、inRNbrで示されるレコードを削除する。

２．５　検索処理と集合演算

HiBaseの特徴である、高速検索と高速集合演算の説明をする。HiBaseではデータベースファイルに対する検索の結果は集合として表現される。さらに、その集合同士の和／積／否定をとることにより、検索結果の集合演算がおこなえる。

```
public int makeSetAll( int inFileNbr ) throws IOException;
```

解説

inFileNbr で示されるデータベースファイル全体の集合を作る。集合番号は、戻り値として返される。

集合内に含まれるレコードの個数を得るには、getSetSize()を用いる。

※HiBaseはデータベース操作の基本を集合演算（集合の四則演算）においている。この集合演算の際に、母集団の存在が操作を容易にすることが多々あるため、本メソッドを用意している。

```
public int makeSetKey( int inFileNbr, byte[] inPacket )
throws IOException;
```

解説

inFileNbr で示されるデータベースファイルから、検索条件にマッチするレコード群の集合を作る。集合番号は、戻り値として返される。

inPacketには、HSearchMakerクラスで構築した検索条件から、HSearchMaker.getPacket()により取り出したバイト配列を指定する。検索条件には、キーに指定されているアイテムのみを指定できる。

集合内に含まれるレコードの個数を得るには、getSetSize()を用いる。

```
public int makeSetItem( int inFileNbr, byte[] inPacket )
throws IOException;
```

解説

inFileNbr で示されるデータベースファイルから、検索条件にマッチするレコード群の集合を作る。集合番号は、戻り値として返される。

inPacketには、HSearchMakerクラスで構築した検索条件から、HSearchMaker.getPacket()により取り出したバイト配列を指定する。検索条件には、キーに指定されていないアイテムを含むことが出来る。

集合内に含まれるレコードの個数を得るには、getSetSize()を用いる。

※makeSetKey()は、キーによる検索条件を受け入れるのに対して、makeSetItem()は、キーでない項目を検索条件式として受け入れる。つまり、HiBaseデータベースファイルの全レコードを読みだし評価することにより集合を作り出す。このため、makeSetItem()は、大容量のファイルに対して実行すると、非常に時間を費やす結果となるため注意を要する。

```
public int makeSetCalc( int inFileNbr, short inOperator,
        int inSetID_A, int inSetID_B ) throws IOException;
```

解説

inFileNbr で示されるデータベースファイルに対して生成された二つの集合を、inOperator が示す条件（HType.oprAnd、HType.oprOr、HType.oprNot）にしたがって、集合演算を行う。演算結果は、新たな集合が生成され、戻り値として集合番号が返される。

inSetID_AとinSetID_B に演算対象となる集合番号を指定する。

演算結果の集合内に含まれるレコードの個数を得るには、getSetSize()を用いる。

```
public int getSetSize( int inFileNbr, int inSetID ) throws IOException;
```

解説

集合内のレコードの個数を得る。

inFileNbr にデータベースファイル番号を、inSetIDに集合番号を指定する。

```
public int getSetRecord( int inFileNbr, int inSetID,
        int inIndex, HRecord outRecord ) throws IOException;
```

解説

集合から、n番目のレコードをとりだす。

inFileNbr で示されるデータベースファイルの、inSetIDが示す集合から、inIndex が示すインデックス番号のレコードを、outRecordに取り出す。戻り値には、そのレコードのレコード番号が返される。

インデックス番号は、0から getSetSize()が返す集合内のレコードの総数－1の範囲で指定できる。

```
public int getSetRNbr( int inFileNbr, int inSetID, int inIndex )
throws IOException;
```

解説

集合から、n番目のレコードのレコード番号を得る。

inFileNbr で示されるデータベースファイルの、inSetIDが示す集合から、inIndex が示すインデックス番号のレコードのレコード番号を、戻り値として返す。

インデックス番号は、0から getSetSize()が返す集合内のレコードの総数－1の範囲で指定できる。

```
public void cancelSet( int inFileNbr, int inSetID ) throws IOException;
```

解説

集合の削除を行う。

inFileNbr で示されるデータベースファイルの、inSetIDが示す集合を削除する。

２．６　検索条件

HDBHandle.makeSetKey()や makeSetItem()に指定する検索条件式を構築するためのクラスについて説明する。

検索条件式の各項を HValueMakerで指定し、各項をつないだ全体の式を HSesrchMakerで構築する。構築後、HSearchMaker.getPacket()で取り出したバイト配列をHDBHandle.makeSetKey()やmakeSetItem()に渡すことになる。

２．６．１　検索条件の構築

```
public      HSearchMaker();
public      HSearchMaker( byte[] inPac );
```

解説

コンストラクタが２種類用意されている。

１つ目のコンストラクタは、デフォルトの、何も条件が指定されていないオブジェクトを生成する。２つ目のコンストラクタは、他の検索条件を初期値として持つオブジェクトを生成する。

```
public  void  clear();
```

解説

内部で保持している検索条件をクリアする。

```
public  void  appendItemExp( short inItemOpr, HValueMaker inValueMaker );
```

解説

inValueMaker で示される条件を、inItemOprで示される演算条件で接合し、追加する。

inItemOprには、以下のいずれかを指定できる。検索条件の最初の項目では、HType.oprBeginを指定しなければならないが、２つ目以降からは任意の演算子を指定できる。

```
HType.oprBegin
HType.oprAnd
HType.oprOr
HType.oprNot
```

```
public  void  setPacket( byte[] inPac );
```

解説

検索条件を設定する。

主に内部利用のためのメソッド。

```
public  byte[]  getPacket();
```

解説

オブジェクト内に構築した検索条件をバイトの配列として取り出す。

取り出したデータは、HDBHandle.makeSetKey()やHDBHandle.makeSetItem()などに渡す検索条件として用いる。

２．６．２　検索条件の１項目の構築

```
public      HValueMaker();
public      HValueMaker( short inItemID, short inValueOpr );
```

解説

コンストラクタが２種類用意されている。

１つ目のコンストラクタは、デフォルトの、何も条件が指定されていないオブジェクトを生成する。２つ目のコンストラクタは、検索条件の初期値を指定できる。

```
public  void  clear();
```

解説

内部で保持している検索条件をクリアする。

```
public  void  setItemOpr( short inItemOpr );
```

解説

項目同士の接続条件を指定する。

実際には、HSearchMaker.appendItemExp()で接続条件を設定するために、このメソッドを使用している。

```
public  void  setOperation( short inItemID, short inValueOpr );
```

解説

検索の対象となるアイテムを inItemID で指定し、inValueOprで値の一致条件を指定する。

inValueOprには、以下の条件を指定できる。

HType.vop_EQ	**完全一致、１つの値を指定**
HType.vop_THR	**範囲指定、２つの値を指定**
HType.vop_EQL	**列挙指定、１つ以上の値を指定**
HType.vop_FMatch	**前方一致、１つの値を指定**
HType.vop_BMatch	**後方一致、１つの値を指定**
HType.vop_AMatch	**中間一致、１つの値を指定**
HType.vop_FContained	**含み前方一致、１つの値を指定**
HType.vop_BContained	**含み後方一致、１つの値を指定**
HType.vop_AContained	**含み中間一致、１つの値を指定**

```
public  void  appendValue( String inValue );
public  void  appendString( String inValue );
public  void  appendNumber( int inValue );
```

解説

検索条件の値を追加する。

値の一致条件によっては、複数個の値を追加しなければならない。たとえば、HType.vop_THR を指定する際には、２つの値を追加する必要がある。

```
public  byte[]  getPacket();
```

解説

オブジェクト内に構築した検索条件をバイトの配列として取り出す。

HSearchMaker.appendItemExp()の内部で利用している。

３．レコードにアクセス　- HRecord

HiBaseへのトランザクション処理は、レコード単位で行われる。HXRecord クラスは HiBaseの内部形式レコードをユーザの扱い易い外部形式に変換する。HiBase API for Java ではJava 言語の String クラスや short 型、int型等をデータベースに保存／読みだしすることが可能だ。

３．１　スキーマ変換クラス

スキーマ変換クラスは、HiBaseの内部形式のレコードをユーザプログラムの扱う外部形式に変換するクラスである。

プログラムではデータをHiBaseに格納するとき、外部形式から内部形式レコードに変換してレコード追加処理をおこなう。逆に、HiBaseに格納されているデータを読みだすときは、レコードの読み出しを行った後、内部形式レコードから外部形式に変換する。

実際には、HRecordクラスを継承している HXRecord クラスを利用することになる。

３．１．１　コンストラクタ

```
public     HRecord();
public     HRecord( int inAllocSize );
public     HRecord( DataInputStream inStream, int inLength );
public     HRecord( HRecord inRecord );
```

解説

HRecordのコンストラクタ。

実際には、これらのコンストラクタを利用することはなく、下記のHXRecord クラスのコンストラクタを利用して、オブジェクトを生成する。

```
public  HXRecord( HDBHandle inDBHdl, int inFileNbr );
public  HXRecord( HDBHandle inDBHdl, int inFileNbr, int inAllocSize );
public  HXRecord( HDBHandle inDBHdl, int inFileNbr,
                     DataInputStream inStream, int inLength );
public  HXRecord( HDBHandle inDBHdl, int inFileNbr, HRecord inRecord );
```

解説

HXRecord のコンストラクタ。

４種類のコンストラクタが用意されている。どのコンストラクタでも、inDBHdlで HDBHandle オブジェクトを指定し、inFileNbr でデータベースファイル番号を指定する必要がある。１つ目のコンストラクタは、デフォルトのバッファサイズを持ち、何もデータがないオブジェクトを生成する。２つ目のコンストラクタは、バッファサイズをinAllocSizeで指定できる。３つ目めのコンストラクタは、inStreamが示すストリームから、inLengthが示す長さのデータを読込んで、初期値とする。４つ目のコンストラクタは、レコードオブジェクトの複製を生成する。

３．１．２　レコードサイズ、クリア、エラーの取得

```
public int    length();
```

解説

格納されているデータの大きさを返す。

```
public void  clear();
```

解説

格納されているデータをクリアする。

```
public int    error();
```

解説

エラーコードを取得する。エラーコードの詳細は、HError.javaを参照。

```
public boolean isNoErr();
```

解説

エラーチェックをする。戻り値が true のとき、エラーは発生していない。また、戻り値がfalse の時、なんらかのエラーが発生していることを示す。

３．１．３　マルチバリュー関連操作

HiBaseではすべてのアイテムに複数（最大 255 個）の値を格納することが出来る。アイテムをマルチバリューにするために特別な宣言は必要ない。また、キーに指定されているアイテムに複数の値を格納（マルチバリュー）しても、それぞれの値に対してキーが生成される。

レコードから値を取り出すときには、対象となるアイテムがマルチバリューなのであれば、何番目の値を取り出すのかを指定する必要がある。このインデックス値は０～２５５までで、対象となるアイテムにいくつの値が入っているのかを事前に調べることができる。

```
public short valueCount( short inItemID );
```

解説

inItemID で示されるアイテムの値がいくつ格納されているかを調べる。

```
public int   valueSize( short inItemID, short inItemIndex );
```

解説

inItemID で示されるアイテムの inItemIndex番目の値の大きさを調べる。

※マルチバリューに関して詳しくは、「HiBaseプログラミングガイド」または、「HiBaseプログラミングリファレンス」を参照。

```
public short valueType( short inItemID, short inItemIndex );
```

解説

inItemID で示されるアイテムの inItemIndex番目の値の種類（タイプ）を調べる。

現在、以下の７種類が用意されている。

HType.hitDate	**日付**
HType.hitTime	**時間**
HType.hitDateTime	**日付＆時間**
HType.hitString	**文字列（256 バイト以下）**
HType.hitNumber	**数字（256 バイト以下）**
HType.hitInteger	**数値**
HType.hitBinary	**バイナリ**

※マルチバリューに関して詳しくは、「HiBaseプログラミングガイド」または、「HiBaseプログラミングリファレンス」を参照。

３．１．４　バリューの追加

```
public void  appendItem( short inItemID, short inDataType,
                                          String inStr );
public void  appendString( short inItemID, String inStr );
public void  appendShort( short inItemID, short inValue );
public void  appendInt( short inItemID, int inValue );
public void  appendBoolean( short inItemID, boolean inFlag );
public void  appendBinary( short inItemID, byte[] inArray );
```

解説

HRecord(HXRecord)オブジェクト中に、値を追加する。

inItemID で対象となるアイテム番号を指定する。同一のアイテム番号に対して値の追加を繰り返すと、マルチバリューとして格納される。

追加する値の種類によって、上記６種類のメソッドのいずれかを利用することになる。

※マルチバリューに関して詳しくは、「HiBaseプログラミングガイド」または、「HiBaseプログラミングリファレンス」を参照。

３．１．５　バリューの取り出し

```
public String   fetchItem( short inItemID, short inItemIndex,
                                          short inDataType );
public String   fetchString( short inItemID, short inItemIndex );
public short    fetchShort( short inItemID, short inItemIndex );
public int      fetchInt( short inItemID, short inItemIndex );
public boolean  fetchBoolean( short inItemID, short inItemIndex );
public byte[]   fetchBinary( short inItemID, short inItemIndex );
```

解説

HRecord(HXRecord)オブジェクト中の値を読みだす。

inItemID で読みだすアイテムの番号を指定し、inItemIndexでマルチバリューの何番目かを指定する。inItemIndexが０の時、一番目の値が読みだされる。

読みだす値の種類によって、上記６種類のメソッドのいずれかを利用することになる。

※マルチバリューに関して詳しくは、「HiBaseプログラミングガイド」または、「HiBaseプログラミングリファレンス」を参照。

３．１．６　バリューの削除

```
public void  deleteItem( short inItemID, short inItemIndex );
public void  deleteItem( short inItemID );
```

解説

HRecord(HXRecord)オブジェクト中の値を削除する。

inItemID で削除するアイテムの番号を指定し、inItemIndexでその何番目の値を削除するのかを指定する。inItemIndexはマルチバリューの並びに対するインデックスなので、複数の値を削除する場合には、インデックスの大きいほうから削除していかねばならない。

２つ目のメソッドでは、そのアイテム中のすべての値を削除する。

レコード中のすべての値を削除する際には、clear()を実行する。

※マルチバリューに関して詳しくは、「HiBaseプログラミングガイド」または、「HiBaseプログラミングリファレンス」を参照。

４．仮想テーブル　－　HVTable

HiBaseではデータベースに対する検索に集合を利用している。検索結果は集合となり、集合同士の演算により実際に検索を行わずとも複雑な検索条件から新たな集合を得ることが出来る。

この集合は、行を HiBaseのレコード、列をHiBase のアイテムからなる表（テーブル）と考えることができる。集合を背景とするこのテーブルを仮想テーブル（virtual table）と呼び、HiBase の集合をより使いやすくし、機能の拡張をしている。

４．１　HVTableList クラス

複数の仮想テーブルを管理する。このクラスにより、仮想テーブルをデータベースに保存／復元することが出来る。

```
public HVTableList( HDBHandle inDBHdl, int inFileNbr );
```

解説

HVTableListクラスのコンストラクタ。

inDBHdl にHDBHandle オブジェクトを指定し、inFileNbr に対象となるデータベースファイルのファイル番号を指定する。

```
public int error();
```

解説

エラーコードを得る。

エラーコードの詳細は、HError.javaを参照。

```
public boolean isNoErr();
```

解説

エラーが発生したかどうかを調べる。

戻り値が true のときエラーは発生していないことを示し、falseのときはエラーが起きたことを示す。

```
public void setDirty( boolean dirty );
```

解説

HVTableListに変更操作があったかどうかのフラグをセットする。

```
public boolean isDirty();
```

解説

変更フラグを調べる。

戻り値が true のときは、このオブジェクトになんらかの変更があったことを示し、falseのときは変更がないことを示す。

戻り値が trueのときは、下記の doSave()を実行し、仮想テーブルをデータベースに保存する必要があることを示している。

テーブルのセーブ／ロード／削除

```
public void doSave( String inUserName );
public void doLoad( String inUserName );
public void doDelete( String inUserName );
```

解説

HVTableListが管理しているHVTable をデータベースに保存したり、データベースから復元したり、データベースに格納されている HVTable を削除することが出来る。

inUserName には任意の名称を指定することができる。保存後、HiBaseを終了し、再び開始したときでも、同じ名前を指定することにより、仮想テーブルを再構築することが出来る。

テーブル名の作成

```
public String makeUniqueName();
```

解説

新たな仮想テーブルの名称を作成し、戻り値として返す。

この名前は、"@VTxxx"（xxxは数字）の形式で、検索対象のデータベースファイルに対してユニークな名前である。

テーブルの作成

```
public HVTable createVTable( String inName, String search );
```

解説

新たな HVTable オブジェクトを生成する。

作成されたテーブルは自動的にリストに追加される。

inName にはテーブルの名称を指定。nullのとき内部的にユニークな名称が設定される。

searchには検索条件式を指定。

検索条件の操作

```
public int getCounts();
```

解説

現在、リストに登録されているHVTable オブジェクトの個数を返す。

```
public void append( HVTable inVTable );
```

解説

inVTable で示すHVTable オブジェクトをリストに追加する。

```
public HVTable fetch( String inName );
```

解説

inName で示す HVTable オブジェクトをリスト中から検索し戻り値として返す。

```
public HVTable fetch( int inIndex );
```

解説

inIndex で示すインデックス番目のHVTable オブジェクトを戻り値として返す。

```
public void remove( HVTable inVTable );
```

解説

inVTable で示すHVTable オブジェクトをリストから削除する。

```
public void remove( String inName );
```

解説

inName で示す HVTable オブジェクトをリストから検索し削除する。

```
public void remove( int inIndex );
```

解説

inIndex で示すインデックス番目のHVTable オブジェクトをリストから削除する。

```
public void removeAll();
```

解説

リスト中のすべての HVTable オブジェクトを削除する。

４．２　HVTable クラス

HVTable クラスは指定された検索条件に一致するレコード群からなる集合を作り出し、テーブル名やインデックス番号によるレコード操作機能を提供する。

```
public HVTable( HVTableList inMother, String inName, String inSearch );
```

解説

HVTable クラスのコンストラクタ。

inMother にHVTableListオブジェクトを指定。inName にこの仮想テーブルの名前を指定。inSearch に検索条件式を指定。

```
public int error();
```

解説

エラーコードを得る。

エラーコードの詳細は、HError.javaを参照。

```
public boolean isNoErr();
```

解説

エラーが発生したかどうかを調べる。

戻り値が true のときエラーは発生していないことを示し、falseのときはエラーが起きたことを示す。

```
public  void setDirty( boolean dirty );
```

解説

変更フラグを設定する。

```
public  boolean isDirty();
```

解説

このオブジェクトに変更があったかどうかを調べる。

戻り値が true のとき変更があったことを示す。

```
public  boolean isMatch( String name );
```

解説

nameで示される名前が同じかどうかを調べる。

戻り値が true のとき名前が一致していることを示す。

```
public  String getName();
public  void setName( String name );
```

解説

仮想テーブルの名称を読みだし／設定する。

```
public  String getSearch();
public  String getSearch( boolean inFromList );
```

解説

検索条件式を読みだす。

inFromListにtrue を指定すると、内部的にリスト表現されているオブジェクトから、検索条件式を生成し、これを返す。

```
public  void setSearch( String inSearch );
public  void setSearch( String inSearch, boolean inToList );
```

解説

検索条件式を設定する。

inSearch に検索条件式を指定する。inToListにtrue を指定すると、検索条件式から、内部表現へ展開する。

```
public  int getSetID();
```

解説

仮想テーブルが内部的に使用している集合の集合番号を返す。

```
public  int getSize();
```

解説

仮想テーブルの行数を返す。

```
public  boolean isActive();
```

解説

集合が生成されているかどうかを調べる。生成されている場合、true を返す。

```
public  int getRNbr( int inIndex ) throws IOException;
```

解説

inIndex で示すインデックス番目のレコードのレコード番号を返す。

```
  public  void getRecord( int inIndex, HRecord outRecord ) throws
IOException;
```

解説

inIndex が示すインデックス番目のレコードをoutRecordが示す HRecordオブジェクトに読みだす。

```
  public  void insertRecord( HRecord inRecord, boolean intoTableToo )
throws IOException;
```

解説

inRecordで示す HRecordオブジェクトをデータベースに追加する。

intoTableToo がtrue の時、仮想テーブルにも追加される。

```
  public  void deleteRecord( int inIndex, boolean inFromTableToo ) throws
IOException;
```

解説

inIndex が示すインデックス番目のレコードをデータベースから削除する。

inFromTableTooが trueの時、仮想テーブルからも削除する。

```
  public  void updateRecord( int inIndex, HRecord inRecord ) throws
IOException;
```

解説

inIndex が示すインデックス番目のレコードをinRecordが示す HRecordオブジェクトの内容で更新する。

検索条件メンバ操作

HVTable では検索条件を文字列形式と内部表現のオブジェクト形式との２通りの表現を持っている。この双方の表現は変換可能で、データベースへの検索条件式の保存時には文字列で行い、検索時にはオブジェクト形式を利用するようになっている。

```
public  void searchToList();
```

解説

検索条件式から内部表現のオブジェクト形式のリストを生成する。

```
public  void searchFromList();
```

解説

内部表現のオブジェクト形式のリストから検索条件式を生成する。

```
public  int getMemberCounts();
```

解説

検索条件の内部表現オブジェクトリストのメンバの個数を返す。

```
public  void appendMember( HVMember inVMember );
public  void appendMember( HVMember inVMember, int inIndex );
```

解説

検索条件メンバオブジェクトを追加する。

inIndex にインデックス番号を指定すると、その位置に追加される。

```
public  HVMember fetchMember( int inIndex );
```

解説

inIndex で指定されるインデックス番目の検索条件メンバオブジェクトを返す。

```
public  void removeMember( HVMember inVMember );
public  void removeMember( int inIndex );
```

解説

検索条件メンバオブジェクトをメンバリストから削除する。

inVMember にHVMemverオブジェクトを指定すると、そのオブジェクトを削除する。

inIndex にインデックス番号を指定すると、そのインデックス番目のオブジェクトを削除する。

仮想テーブルの作成／消去

```
public  void doScan() throws IOException;
```

解説

現在設定されている検索条件により、データベースに対して検索をおこない、仮想テーブルを作成する。

検索後は、isActive()がtrueを返す。

```
public  void doCancel() throws IOException;
```

解説

作成されている仮想テーブルを消去する。

実行後は、isActive()がfalseを返す。

検索条件式のフォーマット

検索条件式の文字列フォーマットを説明する。

検索条件式は、複数の条件をand/or/notでつなぐことにより複数の条件を組み合わせて複雑な検索を行うことが出来る。

<> は省略可能項目を表し、{}はそのうちのいずれか一つを選択することを現す。また、... は繰り返しを現している。

フォーマット：＝条件<{and ¦ or ¦ not} 条件 <...> >;
条件：＝{名前 ¦ 番号} = {値指定 ¦ {テーブル名 ¦ 全て}}

名前：＝文字列（アイテム名、キー名）
番号：＝[<{K ¦ I}>:項目番号]
項目番号：＝数字（アイテム番号、キー番号）
テーブル名：＝文字列
全て：＝[all]

値指定：＝{完全一致 ¦ 前方一致 ¦ 後方一致 ¦ 中間一致 ¦ 範囲 ¦ 列挙}
完全一致：＝ 値
前方一致：＝ 値?
後方一致：＝ ?値
中間一致：＝ ?値?
範囲：＝ 値 - 値
列挙：＝ 値<, 値<...>>

値：＝{数字 ¦ "文字列 "}

また、以下にフォーマットの例を示し説明する。

例１：[K:1] = "山田"?

解説

キー番号１が「山田」で始るレコードを検索する。

ここで１番キーは顧客名簿の姓に対するキーであると仮定する。

例２：[all] not [K:1] = "山田"

解説

キー番号１が「山田」ではないレコードを検索する。

ここで１番キーは顧客名簿の姓に対するキーであると仮定する。

例３：[趣味] = " 映画", " アウトドア " and [性別] = "女 "

解説

趣味が「映画」か「アウトドア」でかつ性別が「女性」であるレコードを検索する。

ここで趣味という名称の付いたキーと性別という名称の付いたアイテム（もしくはキー）があると仮定する。

キーやアイテムの番号を指定するのではなく、その名称で検索条件を指定できる。

仮想テーブル検索条件式の操作

仮想テーブルでは、検索条件式を文字列とメンバオブジェクトの２種類の方法で指定できる。文字列での指定は、上記で説明している。ここでは、メンバオブジェクトを用いての検索条件式の指定方法を説明する。

```
public HVMember( HVTable inMother );
public HVMember( HVTable inMother, short inItemOpr );
```

解説

検索条件オブジェクトの基本クラス。

通常、このクラスを直接生成することはない。

```
public void setDirty( boolean inDirty );
public boolean isDirty();
```

解説

変更があったかどうかのフラグを設定／読み出しをおこなう。

```
public void setItemOperator( short inOpr );
public short getItemOperator();
```

解説

メンバ同士の演算子を設定／取得する。

演算子には、

HType.oprBegin	**開始**
HType.oprAnd	**積**
HType.oprOr	**和**
HType.oprNot	**否定（差）**

を指定する。

メンバの最初の項目には、oprBeginを指定する必要があるが、それ以降のメンバには、任意の演算子を指定でき、前の項目に対する演算をおこなう。

```
public HVSearchMember( HVTable inMother );
```

解説

条件クラスのコンストラクタ。

このクラスを直接生成することはない。通常、HVItemMember クラスまたは、HVKeyMember クラスを生成する。

```
public short getID();
public void setID( short id );
```

解説

条件の対象となるアイテム／キーの番号を設定／取得する。

```
public short getType();
public void setType( short inDataType);
```

解説

条件の対象アイテム／キーのデータ型を設定／取得する。

```
public short getVOper();
public void setVOper( short inValueOpr );
```

解説

条件の対象アイテム／キーと条件の値との一致条件を設定／取得する。

```
public int valueCount();
```

解説

設定されている、値の個数を取得する。

```
public String getValue( int inIndex );
```

解説

inIndex 番目の値を返す。

```
public void addValue( int inIndex, String inValue );
```

解説

inIndex 番目にinValue で示す値を追加する。

```
public void addValueLast( String inValue );
public void addValueFirst( String inValue );
```

解説

inValue で示す値をリストの最初／最後に追加する。

```
public void delValue( int inIndex );
```

解説

inIndex 番目の値を削除する。

```
public HVItemMember( HVTable inMother );
```

解説

アイテムによる検索を指定するメンバのコンストラクタ

```
public HVKeyMember( HVTable inMother );
```

解説

キーによる検索を指定するメンバのコンストラクタ

```
public HVSetMember( HVTable inMother );
```

解説

仮想テーブルの生成条件として、他の仮想テーブルを指定するときに用いるコンストラクタ。

```
public void setTarget( HVTable inTarget );
public HVTable getTarget();
```

解説

検索条件としての仮想テーブルを設定／取得する。

```
public  boolean isActive();
```

解説

検索条件としての仮想テーブルがすでに検索済みかどうかを調べる。

検索済みの時、戻り値がtrue を返す。

```
public  String getName();
```

解説

検索条件としての仮想テーブルの名称を戻り値として返す。

```
public  int getSetID();
```

解説

検索条件としての仮想テーブルの集合番号を返す。

```
public HVAllMember( HVTable inMother );
```

解説

全集合を生成するときに用いるコンストラクタ。