

徐新华 著

数据库与 MIDAS 编程技术



C++ Builder 4
高级编程丛书



清华大学出版社

<http://www.tup.tsinghua.edu.cn>



TP311.13
86



00011420

C++Builder 4 高级编程丛书

数据库与 MIDAS 编程技术

J5 02/04

徐新华 著



C0487431

清华大学出版社

(京)新登字 158 号

内 容 简 介

数据库开发是网络应用中的一项重要内容,尤其是多层客户/服务器体系的开发更是目前非常流行的技术,而 Inprise 公司推出的 C++Builder 4 在这两个方面都提供了很强的支持。本书从 C++Builder 4 数据集公共基类的介绍开始,讲解了如何建立数据访问链路、如何显示数据库中的数据、如何制作 Quick-Report 报表和 TeeChart 图表、如何自定义数据集,以及如何使用数据库浏览器等内容。此外,书中还详细介绍了 C++Builder 4 中的 MIDAS 编程技术。

本书内容翔实,叙述简洁,为读者进一步掌握 C++Builder 4 的数据库开发技术提供了很好的指导,也可作为程序员编程时的参考用书。

版权所有,翻印必究。

本书封面贴有清华大学出版社激光防伪标签,无标签者不得销售。

书 名: 数据库与 MIDAS 编程技术
作 者: 徐新华 著
出 版 者: 清华大学出版社(北京清华大学学研楼,邮编 100084)
<http://www.tup.tsinghua.edu.cn>
责任编辑: 刘彤 汤斌浩
印 刷 者: 北京市人民文学印刷厂
发 行 者: 新华书店总店北京发行所
开 本: 787 × 1092 1/16 印张: 15.75 字数: 364 千字
版 次: 2000 年 5 月第 1 版 2000 年 5 月第 1 次印刷
书 号: ISBN 7-302-03835-X/TP·2242
印 数: 0001 ~ 6000
定 价: 22.00 元

前 言

C++Builder 4 是 Inprise 公司(原为 Borland 公司)1999 年第二季度推出的拳头产品,是当今世界上最优秀的 C++ 和分布式应用程序开发工具。C++Builder 4 最接近 ISO 的 C++ 标准;同时支持 COM 与 CORBA 两大分布式计算规范;内建了全球 ORB 分发数量最多的 VisiBroker 3.3,集成了 CORBA IDL 编译器;进一步兼容了 Microsoft Visual C++ 的代码;增强了远程调试功能;提供了访问 Oracle 8、Microsoft SQL Server 7、Informix 9、Sybase R11、IBM DB/2 Universal Server、InterBase 5.5 等企业级数据库的原生驱动程序;完全支持 Microsoft Transaction Server;内含最新的 MIDAS 2,同时支持 CORBA IIOP、DCOM、DCE RPC 及 TCP/IP 等多种连接方式;支持 CGI、WIN-CGI、ISAPI 及 NSAPI,可以轻松地开发基于 Web 的应用程序。

为了帮助用户全面、准确地掌握 C++Builder 4 的编程思想和用法,我们组织编写了这套《C++Builder 4 高级编程丛书》,主要针对那些已初步掌握了 C++Builder 4 的基本用法,但现在需要进一步精通和提高的读者。本丛书紧紧把握住 C++Builder 4 的基本特征——面向对象,重点从类这一层次把编程思想讲透。只要深刻领会了面向对象的编程思想,那些看上去高深莫测的领域,如 COM、ActiveX、CORBA、MIDAS,也就很容易理解了。

本套丛书分为四册,第一册是《集成开发环境与面向对象编程技术》,第二册是《图形用户界面编程技术》,第三册是《数据库与 MIDAS 编程技术》,第四册是《COM、CORBA 与 Internet 编程技术》。

由于我们的水平有限,再加上时间紧迫,因此尽管做了严格的审核和测试,书中可能还是难免有一些错误,敬请广大读者不吝赐教,谨在此表示感谢。

为了帮助广大程序员更好地掌握这个优秀的开发工具,我们愿意为购买此书的读者提供免费的技术咨询。

北京东大阿尔发软件技术有限公司

地址:北京市上地信息产业基地上地村路 1 号(100085)

电话:(010)62987260 传真:(010)62985141

网址:<http://www.allfa.com.cn> 邮件:books@allfa.com.cn

第一章 数 据 集

数据集是一个抽象而又具体的概念,说它抽象,是因为它并不直接描述数据库的任何记录和字段。说它具体,是因为在 C++Builder 4 中数据集有三种确切的表现形式——表、查询、存储过程,这三种形式的数据集分别用 TTable、TQuery、TStoredProc 来操纵。

TTable、TQuery、TStoredProc 的直接上级是 TDBDataSet,而 TDBDataSet 的上级是 TBDEDataSet, TBDEDataSet 又是从 TDataSet 继承下来的。

C++Builder 4 引入了分布式数据集的概念,并且用 TClientDataSet 来实现并操纵分布式数据集,而 TClientDataSet 也是从 TDataSet 继承下来的。

由此可见,不管是传统的基于 BDE 的数据集,还是分布式数据集, TDataSet 是它们的共同基类。它们之间的继承关系如图 1.1 所示。

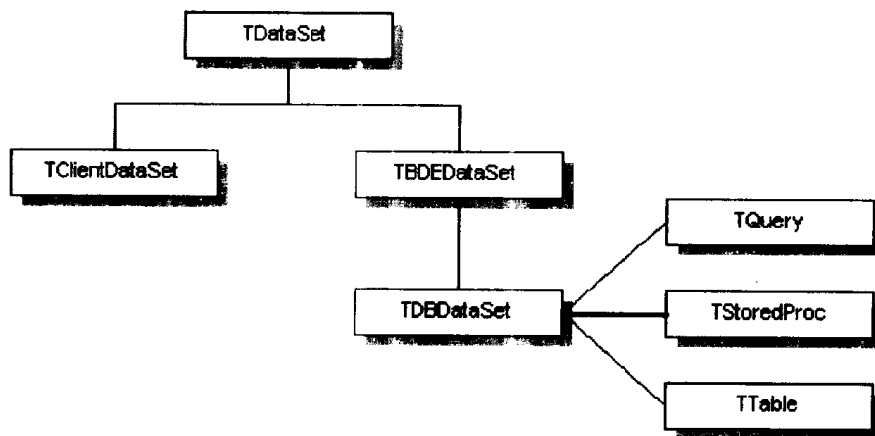


图 1.1 数据集的继承关系

1.1 TDataSet

TDataSet 是所有数据集的虚拟基类。不能直接创建它的对象实例,也不能直接访问它的特性和方法,因为这些特性和方法大部分是虚拟的或抽象的。

如果从功能上划分, TDataSet 的成员可以分为这么几大块: 打开和关闭数据集、浏览记录、编辑数据、书签管理、控制连接、访问字段、记录缓冲区管理、过滤、事件。

Active 特性

声明: `_property bool Active;`

如果这个特性设为 true, 相当于调用 Open 函数打开数据集。如果这个特性设为 false, 相当于调用 Close 函数关闭数据集。

如果修改了数据集的某个记录,并在还没有 Post 之前就把 Active 特性设为 false,则该修改将作废。因此,建议您响应 BeforeClose 事件,在数据集变成非活动状态之前使修改有效。

也可以读 Active 特性来判断数据集当前是否打开,程序示例如下:

```
void _fastcall TForm1::Button1Click(TObject * Sender)
{
    Table1 -> Active =! (Table1 -> Active);
    if(Table1 -> Active)
        Caption = "表已打开";
    else
        Caption = "表还没有打开";
}
```

AutoCalcFields 特性

声明: `_property bool AutoCalcFields;`

当程序从数据集中检索记录时,将触发 OnCalcFields 事件。如果 AutoCalcFields 特性设为 true,当程序修改了数据集的某个字段时,也将触发这个事件。

注意:如果 AutoCalcFields 设为 true,在处理 OnCalcFields 事件的句柄中不能再修改数据集,否则将老是触发这个事件,可能变成死循环。

BlockReadSize 特性

声明: `_property int BlockReadSize;`

如果这个特性设为 0,程序每调用一次 Next,相应的数据控件将刷新一次。为了提高应用程序的性能,可以把 BlockReadSize 特性设为一个大于 0 的数,当程序调用 Next 若干次后,相应的数据控件才刷新一次。

Bof 特性

声明: `_property bool Bof;`

如果当前记录是数据集的第一条记录,这个只读的特性就返回 true。当程序第一次打开一个数据集或者调用了 First 函数,当前记录就是第一条记录。程序示例如下:

```
void _fastcall TForm1::Button1Click(TObject * Sender)
{
    Table1 -> First();
    if(Table1 -> Bof)
        ShowMessage("当前是在表的开头");
}
```

Bookmark 特性

声明: `_property System::AnsiString Bookmark;`

书签用于标记数据集某个记录位置,以后就可以快速地回到书签所标记的地方。

如果读这个特性的话,这个特性返回当前记录的书签(如果有的话)。如果写这个特性,通常用于指定一个已有的书签,书签标注的记录就变成当前记录。

CanModify 特性

声明: `_property bool CanModify;`

如果这个只读的特性返回 `false`, 表示数据集的数据不能修改。不过, 即使 `CanModify` 特性返回 `true`, 也并不意味着数据集一定能修改, 因为 SQL 表还有个访问权限的问题。

千万不要把 `ReadOnly` 特性与 `CanModify` 特性混淆起来。如果 `ReadOnly` 特性设为 `true`, `CanModify` 特性肯定返回 `false`。但如果 `ReadOnly` 特性设为 `false`, 只有当获得了对数据集的读写权限后, `CanModify` 特性才返回 `true`。

DefaultFields 特性

声明: `_property bool DefaultFields;`

如果数据集中包含运行期动态生成的字段 (`TField` 对象), 这个只读的特性就返回 `true`。如果用字段编辑器创建了永久的字段, 这个特性就返回 `false`。

Eof 特性

声明: `_property bool Eof;`

如果当前记录是数据集的最后一记录, 这个只读的特性就返回 `true`。当程序打开一个空的数据集或者调用了 `Last` 函数, 当前记录就是最后一记录。程序示例如下:

```
void _fastcall TForm1::DataSource1DataChange(TObject * Sender, TField * Field)
{
    StatusBar1->SimplePanel = true;
    if(Table1->Eof)
        StatusBar1->SimpleText = "现在是在表的最后";
}
```

FieldCount 特性

声明: `_property int FieldCount;`

这个只读的特性返回数据集中的字段数。由于数据集中可能包含动态生成的字段, 因此, 这个特性返回的字段数与数据集物理存储时的字段数可能不同。程序示例如下:

```
void _fastcall TForm1::Button1Click(TObject * Sender)
{
    String Info("表中的字段有:");
    for(int i = 0; i < Table1->FieldCount; i++)
        Info = Info + Table1->Fields[i]->FieldName + ", ";
    ShowMessage(Info);
}
```

FieldDefs 特性

声明: `_property TFieldDefs * FieldDefs;`

C++Builder 4 用 `TFieldDef` 对象来定义数据集中的每一个字段(计算字段除外)。

一个数据集中有几个字段, 就需要有几个 `TFieldDef` 对象。这些 `TFieldDef` 对象由 `FieldDefs` 特性返回的 `TFieldDefs` 对象来管理。

一般情况下, 不需要修改字段的定义, 除非要在运行期动态创建一个表或者字段时才需要给出字段的定义。下面的程序示例创建了一个表:

```

void _fastcall TForm1::Button1Click(TObject * Sender)

    Table1 -> Active = false;

    Table1 -> DatabaseName = "BCDEMOS";           //先描述表的类型和名称
    Table1 -> TableType = ttParadox;
    Table1 -> TableName = "CustInfo";

    Table1 -> FieldDefs -> Clear();              //给出字段的定义
    Table1 -> FieldDefs -> Add("Field1", ftInteger, 0, true);
    Table1 -> FieldDefs -> Add("Field2", ftString, 30, false);

    Table1 -> IndexDefs -> Clear();              //描述表的索引
    Table1 -> IndexDefs -> Add("", "Field1", TIndexOptions() << ixPrimary << ixU-
    nique);
    Table1 -> IndexDefs -> Add("Fld2Index", "Field2",
    TIndexOptions() << ixCaseInsensitive);

    Table1 -> CreateTable();                      //创建这个表
}

```

Fields 特性

声明: `_property TField * Fields[int Index];`

通过这个特性,可以访问数据集中的每一个字段(TField 对象),序号从 0 开始。

一般来说,要访问字段的值最好通过 FieldValues 特性,不过,如果知道字段的数据类型的话,也可以通过 Fields 特性来访问字段的值,例如:

```
Edit1 -> Text = Table1 -> Fields[0] -> AsString;
```

上面这行代码用于读字段的值。如果要对字段赋值,可以参考下面的例子:

```

Table1 -> Edit();
Table1 -> Fields[0] -> AsString = Edit1 -> Text;
Table1 -> Post();

```

Fields 特性往往与 FieldCount 特性配合起来使用,程序示例如下:

```

void _fastcall TForm1::Button1Click(TObject * Sender)
{
    for(int i = 0; i < Table1 -> FieldCount; i++)
        ListBox1 -> Items -> Add(Table1 -> Fields[i] -> FieldName);
}

```

FieldValues 特性

声明: `_property System::Variant FieldValues[System::AnsiString FieldName];`

通过这个特性可以按字段名来访问字段的值。程序示例如下:

```

Customers -> Edit();
Customers -> FieldValues["CustNo"] = Edit1 -> Text;
Customers -> Post();

```

由于 FieldValues 是 TDataSet 的默认特性,因此上述程序可以简化为:


```
Customers -> Edit();
Customers["CustNo"] = Edit1 -> Text;
Customers -> Post();
```

由于 FieldValues 特性的数据类型是 Variant,它可以表达任何数据类型的值,因此,不需要用诸如 AsString、AsInteger 来转换字段的值。

Filtered 特性

声明: `_property bool Filtered;`

如果这个特性设为 true,表示将对数据集中的记录进行过滤,过滤条件由 Filter 特性设置,过滤时将触发 OnFilterRecord 事件;如果这个特性设为 false,表示不进行过滤。

为什么要使用过滤?因为程序关心的往往只是数据集中满足特定条件的部分记录,这就需把不满足条件的记录过滤掉。要过滤记录,有两种方式:一是在运行期调用 Locate 过程寻找匹配的记录;二是把 Filtered 特性设为 true,这样,数据集的每条记录都会产生 OnFilterRecord 事件。在处理 OnFilterRecord 事件的句柄中,把 Accept 参数设为 false,表示过滤掉该记录。例如,要只显示 State 字段的值是“CA”的记录,程序示例如下:

```
void _fastcall TForm1::Table1FilterRecord(TDataSet * DataSet, bool &Accept)
{
    Accept = DataSet["State"] == "CA";
}
```

在运行期,可以把 Filtered 特性设为 false 从而使 OnFilterRecord 事件不再触发,也就是说不进行过滤。不过,如果要显示所有的记录,程序必须显式地调用 Refresh。

可以在运行期给 OnFilterRecord 事件重新指定一个事件句柄,从而使程序按不同的过滤条件对记录进行过滤。程序示例如下:

```
Table1 -> OnFilterRecord = AnotherFilterRecord;
```

最后要说明的是,尽管使用过滤能够只显示一部分记录,但如果数据集中记录很多,最好还是使用查询或者设置范围。

Filter 特性

声明: `_property System::AnsiString Filter;`

这个特性用于设置过滤条件。过滤有点类似于查询,用于从数据集中选取一些满足特定条件的记录,当然,查询的功能要强大得多。

Filter 特性是字符串,其格式非常类似于 SQL 语句的 WHERE 部分,可以使用比较操作符,包括 <、>、>=、<=、=、<> 等。程序示例如下:

```
OrderNum >= 16;
```

表示只选取 OrderNum 字段的值大于等于 16 的记录。

如果需要指定多重条件,可以使用关系操作符(包括 AND、NOT、OR 等)把多个条件连接起来。程序示例如下:

```
(OrderNum >= 16) AND (Sex = "Man");
```

表示只选取 OrderNum 字段的值大于等于 16 并且 Sex 字段为“Man”的记录。
如果字段的名称本身含有空格,要用一对方括号把字段名括起来,示例如下:

```
[Home State] = "CA" OR [Home State] = "MA"
```

FilterOptions 特性

声明: `_property TFilterOptions FilterOptions;`

这个特性设置过滤的选项, TFilterOptions 是一个集合,可以包含下列元素:

- foCaseInsensitive 大小写不敏感。
- foNoPartialCompare 对于字符串类型的字段必须全字匹配,不允许部分匹配。

Found 特性

声明: `_property bool Found`

如果这个只读的特性返回 true,表示 FindFirst、FindLast、FindNext、FindPrior 等函数调用成功。

Modified 特性

声明: `_property bool Modified;`

这是个运行期只读的特性,如果返回 true,表示当前记录中某个字段被修改了,但还没有被写到数据集中,当程序调用 Cancel 或 Post 后,这个特性又恢复为 false。程序示例如下:

```
if (Table1 -> Modified)
{
    if (MessageDlg("要保存当前记录吗?", mtConfirmation, TMsgDlgButtons()
        << bYes << mbNo, 0) = mrYes)
        Table1 -> Post();
    else Table1 -> Cancel();
}
```

NestedDataSets 特性

声明: `_property Classes::TList * NestedDataSets;`

C++Builder 4 支持 Oracle8 的嵌套数据集功能。这个特性将返回一个列表,该列表列出了数据集的所有嵌套数据集,也称子数据集。

ObjectView 特性

声明: `_property bool ObjectView;`

如果这个特性设为 true,数据集中的字段按照字段之间的继承关系以树状结构存储。如果这个特性设为 false,数据集中的字段按照它们在 Fields 数组中的顺序线性存储。

RecordCount 特性

声明: `_property int RecordCount;`

这个只读的特性返回数据集的记录数,注意: RecordCount 特性返回的记录数与数据集物理存储的记录数可能不同,因为有可能使用了过滤。不过,对于 dBASE 表来说, RecordCount 特性返回的总是数据集物理存储的全部记录数,即使调用了 SetRange 函数。

下面用一个百分数来显示处理数据集记录的进度,程序示例如下:

```
int I = 1;
Table1 -> First();
While (! Table1 -> Eof)
{
    StatusBar1 -> Panels[0] -> Text = IntToStr((I * 100) div RecordCount);
    I += 1;
    Table1 -> Next();
}
```

RecordSize 特性

声明: `_property Word RecordSize;`

这个只读的特性返回当前记录缓冲区的长度(以字节为单位)。

State 特性

声明: `_property TDataSetState State;`

这个只读的特性返回数据集当前的状态。可以是以下值:

- `dsInactive` 数据集已关闭,不能访问;
- `dsBrowse` 数据集处于活动状态,能够浏览它的数据但不能编辑;
- `dsEdit` 数据集处于活动状态,能够编辑;
- `dsInsert` 数据集处于活动状态,能够插入记录;
- `dsSetKey` 数据集处于活动状态,正在调用 `SetRange`;
- `dsCalcFields` 数据集处于活动状态,正在处理 `OnCalcFields` 事件;
- `DsFilter` 数据集处于活动状态,正在处理 `OnFilterRecord` 事件。

```
void _fastcall TForm1::Button1Click(TObject * Sender)
{
    TBlobStream * Stream1 = new TBlobStream(Table1Notes, bmReadWrite);
    try
    {
        Table1 -> Edit();
        if (Table1 -> State == dsEdit)
        {
            Stream1 -> Seek(60, 0);
            Stream1 -> Truncate();
            Table1 -> Post();
        }
    }
    catch (...)
    {
        delete Stream1;
        throw;
    }
    delete Stream1;
}
```

ActiveBuffer 函数

声明: `char * _fastcall ActiveBuffer(void);`

这个函数返回指向当前记录缓冲区的指针。

Append 函数

声明: `void _fastcall Append(void);`

这个函数在数据集的末尾添加一个新的空记录。在调用这个函数之前最好要访问 CanModify 特性,当 CanModify 特性返回 true 时再调用 Append 函数。

调用了这个函数以后,还必须调用 Post 函数才真正把一条新记录插入到数据集中。如果调用 Cancel 函数将取消刚才的 Append 操作。程序示例如下:

```
void _fastcall TForm1::Button1Click(TObject * Sender)
{
    Table1 -> Append();
    Table1 -> FieldValues["ALPHANUMERIC"] = Edit1 -> Text;
    Table1 -> FieldValues["INTEGER"] = StrToInt(Edit2 -> Text);
    Table1 -> Post();
}
```

对于没有建立索引的 Paradox 表或 dBASE 表来说,新记录添加在数据集的最后。对于建立了索引的 Paradox 表或 dBASE 表来说,新记录将自动移到一个恰当的位置。

AppendRecord 函数

声明: `void _fastcall AppendRecord(const System::TVarRec * Values, const int Values _ Size);`

这个函数与 Append 函数类似,也是用于把一条空记录加到表的末尾。不同的是,调用 Append 函数后还得逐个给字段赋值,而 AppendRecord 函数本身就带一个 Values 参数。Values 参数是一个数组,用于指定记录中字段的值,其元素的顺序、个数、数据类型都必须和字段一致,其他注意事项请见 Append 函数。程序示例如下:

```
Customer -> AppendRecord(ARRAYOFCONST((CustNoEdit -> Text, CoNameEdit -> Text,
    AddrEdit -> Text, Null, Null, Null, Null, Null, Null, DiscountEdit -> Text)));
```

BookmarkValid 函数

声明: `virtual bool _fastcall BookmarkValid(void * Bookmark);`

如果 Bookmark 参数指定的书签是合法的,这个函数就返回 true。

Cancel 函数

声明: `virtual void _fastcall Cancel(void);`

如果修改了当前记录,在没有调用 Post 之前,可以调用 Cancel 函数来取消修改,数据集回到 dsBrowse 状态。

ClearFields 函数

声明: `void _fastcall ClearFields(void);`

这个函数将把当前记录的所有字段的值清空。如果数据集当前不处于 dsInsert 或 dsEdit 状态,调用这个函数将触发异常。ClearFields 函数可能会导致计算字段被更新,并触发与数据集关联的 TDataSource 元件的 OnDataChange 事件。

Close 函数

声明: void _fastcall Close(void);

这个函数用于关闭数据集,使它处于 dsInactive 状态,相当于把 Active 特性设为 false。在修改数据集的其他特性前,一定要先调用 Close 函数关闭数据集。

ControlsDisabled 函数

声明: bool _fastcall ControlsDisabled(void);

如果这个函数返回 true,表示数据集与数据源暂时断开了连接。这时候,即使数据集的数据发生变化,那些数据控件也不会刷新。程序示例如下:

```
void _fastcall TForm1::ReEnableControls(TDataSet * DataSet)
{
    while (DataSet -> ControlsDisabled())
        DataSet -> EnableControls();
}
```

CursorPosChanged 函数

声明: void _fastcall CursorPosChanged(void);

一般情况下,不需要调用这个函数,除非您直接调用了 BDE 的 API,需要调用这个函数通知数据集,当前记录的位置可能有了变化。

Delete 函数

声明: void _fastcall Delete(void);

这个函数删除当前记录,这样,下一条记录就变成当前记录。如果删除的已经是最后一条记录,那么前一条记录变成当前记录。

DisableControls 函数

声明: void _fastcall DisableControls(void);

这个函数用于临时断开数据集与数据源的连接。这样,即使数据集发生变化,相应的数据控件也不会随着刷新。当需要连续多次在记录之间移动时,最好调用这个函数暂时取消与数据控件的连接;等翻滚到所需要的记录时,再调用 EnableControls 函数恢复连接。这样,可以避免数据控件老是不必要的刷新数据。程序示例如下:

```
CustTable -> DisableControls();
try
{
    CustTable -> First();
    while (! CustTable -> Eof)
    {
        // 处理记录
        CustTable -> Next();
    }
}
```

```

    |
    | _finally
    |
    | CustTable -> EnableControls();
    |
    |

```

EnableControls 函数

声明: void _fastcall EnableControls(void);

这个函数用于恢复数据集与数据源的连接。调用 DisableControls 函数将使一个内部的计数器加 1,调用 EnableControls 函数将使这个计数器减 1。当计数器减到 0 的时候,才真正恢复与数据源的连接。因此,如果程序多处调用了 DisableControls 函数,就必须相应地调用多次 EnableControls 函数,才能真正使数据集与数据源的连接得到恢复。

FieldByName 函数

声明: TField * _fastcall FieldByName(const System::AnsiString FieldName);

这个函数通过字段名来访问当前记录的某个字段(TField 对象)。程序示例如下:

```

void _fastcall TForm1::Button1Click(TObject * Sender)
{
    Table1 -> Insert();
    Table1 -> FieldByName("QUANTITY") -> AsInteger = StrToInt(Edit1 -> Text);
    Table1 -> Post();
}

```

用字段名来访问某个字段比用字段的序号来访问某个字段要安全些,因为很有可能记错了序号,或者字段的顺序在运行期发生了改变。

不过,用字段名访问字段应确保数据集中确实有这个字段,如果指定的字段不存在,将触发异常。因此,如果不能确定数据集中是否有这个字段,最好调用 FindField 函数。

FindField 函数

声明: TField * _fastcall FindField(const System::AnsiString FieldName);

这个函数与 FieldByName 函数类似,也是通过字段名访问字段。不同的是,如果指定的字段不存在,函数将返回 NULL 而不会触发异常。程序示例如下:

```

void _fastcall TForm1::Button1Click(TObject * Sender)
{
    DataSource1 -> Edit();
    Table1 -> FindField("CustNo") -> AsString = "1234";
}

```

First 函数

声明: void _fastcall First(void);

这个函数使过滤范围内的第一条记录变成当前记录。如果当前有未决的操作,将首先调用 Post 函数。如果没有设置过滤范围,这个函数将返回表的第一条记录。程序示例如下:

```

void _fastcall TForm1::Button1Click(TObject * Sender)
{
    ProgressBar1 -> Min = 0;
    ProgressBar1 -> Max = Table1 -> RecordCount;
    Table1 -> First();
    for (int i = ProgressBar1 -> Min; i <= ProgressBar1 -> Max; i++)
    {
        ProgressBar1 -> Position = i;
        Table1 -> Next();
        // 处理记录
    }
}

```

Last 函数

声明: void _fastcall Last(void);

这个函数使过滤范围内的最后一条记录变成当前记录。

Prior 函数

声明: void _fastcall Prior(void);

这个函数使过滤范围内的前一条记录变成当前记录。

Next 函数

声明: void _fastcall Next(void);

这个函数使过滤范围内的下一条记录变成当前记录。

GetBookmark 函数

声明: virtual TBookmark * _fastcall GetBookmark(void);

这个函数创建一个书签来标记当前记录,以后就可以调用 GotoBookmark 函数重新回到这个记录。如果数据集是空的或者不在 dsBrowse 状态,函数返回 NULL。程序示例如下:

```

TBookmark MyBookmark;
// -----
_fastcall TForm1::TForm1(TComponent * Owner)
: TForm(Owner)
{
}
// -----
void _fastcall TForm1::Button1Click(TObject * Sender)
{
    MyBookmark = Table1 -> GetBookmark();
    ...
}
// -----
void _fastcall TForm1::Button2Click(TObject * Sender)
{
    Table1 -> GotoBookmark(MyBookmark);
}

```

```
Table1 -> FreeBookmark(MyBookmark);
```

注意 当数据集被关闭或者索引改变,书签将无效。

GotoBookmark 函数

声明: void _fastcall GotoBookmark(void * Bookmark);

这个函数使 Bookmark 参数指定的书签所标记的记录重新成为当前记录。

FreeBookmark 函数

声明: virtual void _fastcall FreeBookmark(void * Bookmark);

书签也是系统资源,用好后应当释放。

GetCurrentRecord 函数

声明: virtual bool _fastcall GetCurrentRecord(char * Buffer);

这个函数把当前记录放到 Buffer 参数指定的缓冲区,缓冲区的大小应不小于 Record-Size 特性的值。如果函数调用成功就返回 true。

GetFieldList 函数

声明: void _fastcall GetFieldList(Classes::TList * List, const AnsiString FieldNames);

这个函数把若干字段(TField 对象)复制到 List 参数指定的列表中。其中,Field-Names 参数指定这些字段的名称,彼此之间用分号隔开。

GetFieldNames 函数

声明: void _fastcall GetFieldNames(Classes::TStrings * List);

这个函数把数据集中所有字段的名称放到 List 参数指定的列表中。

Insert 函数

声明: HIDESBASE void _fastcall Insert(void);

这个函数与 Append 函数相似。不同的是,Insert 函数把新的记录插在当前位置而不是加在数据集的末尾。

InsertRecord 函数

声明: void _fastcall InsertRecord(const System::TVarRec * Values, const int Values _ Size);

这个函数与 AppendRecord 函数相似。不同的是,InsertRecord 把新记录插在当前位置。程序示例如下:

```
Customer -> InsertRecord(ARRAYOFCONST((CustNoEdit -> Text, CoNameEdit -> Text, AddrEdit -> Text, Null, Null, Null, Null, Null, Null, DiscountEdit -> Text)));
```

IsEmpty 函数

声明: bool _fastcall IsEmpty(void);

如果数据集是空的,也就是说一条记录都没有,这个函数就返回 true。

IsLinkedTo 函数

声明: `bool _fastcall IsLinkedTo(TDataSource * DataSource);`

这个函数判断数据集是否与指定的 TDataSource 元件相联,如果相联就返回 true。

IsSequenced 函数

声明: `virtual bool _fastcall IsSequenced(void);`

如果这个函数返回 true,表示数据集的每条记录都对应着一个唯一的序号,这种情况下,用 RecNo 特性就可以遍历所有记录。

Locate 函数

声明: `virtual bool _fastcall Locate(const System::AnsiString KeyFields, const System::Variant &KeyValues, TLocateOptions Options);`

这个函数在数据集中搜索符合特定条件的记录。如找到,就把找到的记录作为当前记录。KeyFields 参数是一个字符串,用于指定要搜索的字段名,字段与字段之间用分号隔开。KeyValues 参数用于指定每个字段相应的值。Options 参数用于设置搜索选项。

下面这个例子在 CustTable 表中搜索 Company 字段的值为“Professional Divers, Ltd.”的记录:

```
void _fastcall TForm1::Button1Click(TObject * Sender)
{
    TLocateOptions SearchOptions;
    SearchOptions = SearchOptions << loPartialKey;
    if(Table1->Locate("Company", "Professional Divers, Ltd.", SearchOptions))
        ShowMessage("Found the Record");
    else
        ShowMessage("Record not Found");
}
```

Lookup 函数

声明: `virtual System::Variant _fastcall Lookup(const System::AnsiString KeyFields, const System::Variant &KeyValues, const System::AnsiString ResultFields);`

这个函数与 Locate 相似,用于从数据集中搜索符合特定条件的记录。KeyFields 参数用于指定要搜索的字段名,字段与字段之间用分号隔开。KeyValues 参数是对应于这些字段的值。ResultFields 参数用于指定如果找到匹配的记录时要返回哪些字段的值。

MoveBy 函数

声明: `int _fastcall MoveBy(int Distance);`

这个函数用于把另一个记录变成当前记录。Distance 参数指定相对于当前记录的距离。例如,MoveBy(-2)表示前面第二个记录变成当前记录,MoveBy(3)表示后面第三个记录变成当前记录,如果数据集处于 dsInsert 或 dsEdit 状态,MoveBy 函数将首先调用 Post。

Open 函数

声明: `void _fastcall Open(void);`