

# 优炫数据库NCI接口使用手册 2.1



**UXSINO**  
优炫软件

---

# 优炫数据库NCI接口使用手册 2.1

版权 © 2016-2023 北京优炫软件股份有限公司

## 法律声明

优炫数据库管理系统(简称: UXDB) 是由北京优炫软件股份有限公司开发并发布的一款商业性数据库管理系统。

优炫数据库管理系统 (UXDB) 的一切知识产权以及与该软件产品相关的所有信息内容, 包括但不限于: 文字表述及其组合、图标、图饰、图表、色彩、界面设计、版面框架、有关数据、及电子文档等均属北京优炫软件股份有限公司所有。本软件及其文档的任何使用、复制、修改、出租、传播、销售及分发等行为均须经北京优炫软件股份有限公司书面许可。

凡侵犯北京优炫软件股份有限公司知识产权的行为, 北京优炫软件股份有限公司将依法追究其法律责任。

本声明的最终解释权归属于北京优炫软件股份有限公司。



和其他优炫公司商标均为北京优炫软件股份有限公司的商标。

本文档提及的其他所有商标或注册商标, 由各自的所有人拥有。

## 注意

由于产品版本安装或其他原因, 本文档内容会不定期进行更新。除非另有约定, 本文档仅作为使用指导, 本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

北京优炫软件股份有限公司 (总部)

- 地址: 北京市海淀区学院南路62号中关村资本大厦11层 (邮编: 100081)
  - 网址: <http://www.uxsino.com>
  - 邮箱: <uxdb\_support@uxsino.com>
  - 电话: 010-82886998
  - 传真: 010-82886338
  - 服务热线: 400-650-7837
-

---

# 目录

前言	ix
1. 文档目的	ix
2. 文档对象	ix
3. 修改记录	ix
1. 概述	1
2. 安装与配置	2
2.1. 安装	2
2.2. 配置	2
3. 接口介绍	4
3.1. 句柄	4
3.1.1. NCIEnvCreate	4
3.1.2. NCIHandleAlloc	5
3.1.3. NCIHandleFree	6
3.1.4. NCIInitialize	7
3.1.5. NCIEnvInit	7
3.1.6. NCIDescriptorAlloc	8
3.1.7. NCIDescriptorFree	9
3.2. 属性	10
3.2.1. NCIAAttrSet	10
3.2.2. NCIParamGet	13
3.2.3. NCIAAttrGet	14
3.2.4. NCIDescribeAny	16
3.2.5. NCIServerVersion	18
3.3. 连接	18
3.3.1. NCIServerAttach	18
3.3.2. NCIServerDetach	19
3.3.3. NCISessionBegin	19
3.3.4. NCISessionEnd	20
3.3.5. NCILogon	21
3.3.6. NCILogoff	22
3.4. 错误	22
3.4.1. NCIErrorGet	22
3.5. 语句	23
3.5.1. NCISmtPrepare	23
3.5.2. NCISmtExecute	24
3.5.3. NCIBindByPos	25
3.5.4. NCIDefineByPos	26
3.5.5. NCISmtFetch	28
3.5.6. NCISmtFetch2	28
3.5.7. NCIBindByName	30
3.5.8. NCIBindArrayOfStruct	31
3.5.9. NCIDefineArrayOfStruct	32
3.5.10. NCISmtSetPieceInfo	33
3.5.11. NCISmtGetPieceInfo	34
3.5.12. NCIBindDynamic	35
3.5.13. NCIDefineDynamic	36
3.6. 事务	37
3.6.1. NCITransCommit	37
3.6.2. NCITransRollback	38
3.6.3. NCITransStart	38
3.6.4. NCITransDetach	39

3.7. 大对象 .....	40
3.7.1. NCILobRead .....	40
3.7.2. NCILobWrite .....	41
3.7.3. NCILobGetLength .....	43
3.7.4. NCILobCreateTemporary .....	43
3.7.5. NCILobAssign .....	44
3.7.6. NCILobFreeTemporary .....	45
3.7.7. NCILobIsEqual .....	45
3.7.8. NCILobEnableBuffering .....	46
3.7.9. NCILobDisableBuffering .....	46
3.7.10. NCILobFlushBuffer .....	47
3.7.11. NCILobAppend .....	48
3.7.12. NCILobTrim .....	48
3.7.13. NCILobErase .....	49
3.7.14. NCILobCopy .....	49
3.7.15. NCILobOpen .....	50
3.7.16. NCILobClose .....	51
3.7.17. NCILobLocatorIsInit .....	51
3.7.18. NCILobLocatorAssign .....	52
3.8. RAW对象 .....	52
3.8.1. NCIRawPtr .....	52
3.8.2. NCIRawSize .....	53
3.8.3. NCIRawAllocSize .....	53
3.8.4. NCIRawAssignBytes .....	54
3.8.5. NCIRawResize .....	54
3.8.6. NCIRawAssignRaw .....	55
3.9. 时间 .....	56
3.9.1. NCIDateTimeConstruct .....	56
3.9.2. NCIDateTimeFromText .....	57
3.9.3. NCIDateTimeToText .....	58
3.9.4. NCIDateTimeGetDate .....	59
3.9.5. NCIDateTimeGetTime .....	59
3.10. 时间间隔 .....	60
3.10.1. NCIIntervalSetYearMonth .....	60
3.10.2. NCIIntervalGetYearMonth .....	61
3.10.3. NCIIntervalSetDaySecond .....	61
3.10.4. NCIIntervalGetDaySecond .....	62
3.10.5. NCIIntervalCompare .....	63
3.10.6. NCIIntervalAdd .....	63
3.10.7. NCIIntervalAssign .....	64
3.10.8. NCIIntervalToText .....	65
3.10.9. NCIIntervalFromText .....	65
3.10.10. NCIIntervalCheck .....	66
3.10.11. NCIIntervalDivide .....	67
3.10.12. NCIIntervalMultiply .....	67
3.10.13. NCIIntervalSubtract .....	68
3.11. 数字 .....	68
3.11.1. NCINumberFromInt .....	68
3.11.2. NCINumberToInt .....	69
3.11.3. NCINumberFromText .....	70
3.11.4. NCINumberToText .....	71
3.11.5. NCINumberFromReal .....	71
3.11.6. NCINumberToReal .....	72
3.11.7. NCINumberAdd .....	73

3.11.8.	NCINumberAssign	73
3.11.9.	NCINumberDiv	74
3.11.10.	NCINumberIsInt	74
3.11.11.	NCINumberMod	75
3.11.12.	NCINumberMul	75
3.11.13.	NCINumbersub	76
3.12.	直接文件操作	77
3.12.1.	NCIDirPathPrepare	77
3.12.2.	NCIDirPathColArrayEntrySet	77
3.12.3.	NCIDirPathColArrayToStream	78
3.12.4.	NCIDirPathLoadStream	79
3.12.5.	NCIDirPathFinish	79
3.13.	字符串操作	80
3.13.1.	NCIStringAllocSize	80
3.13.2.	NCIStringAssign	80
3.13.3.	NCIStringAssignText	81
3.13.4.	NCIStringPtr	82
3.13.5.	NCIStringResize	82
3.13.6.	NCIStringSize	83
4.	附录	84
4.1.	使用示例	84
4.1.1.	编译命令	84
4.1.2.	创建表	84
4.1.3.	插入数据及事务提交	88
4.1.4.	多行绑定	92
4.1.5.	大对象	97
4.1.6.	多事务	101

---

## 表格清单

1. 文档更新记录 .....	ix
2.1. 其他参数 .....	3
3.1. NCIEnvCreate 参数说明 .....	4
3.2. NCIHandleAlloc 参数说明 .....	5
3.3. NCIHandleFree 参数说明 .....	6
3.4. NCIInitialize 参数说明 .....	7
3.5. NCIEnvInit 参数说明 .....	8
3.6. NCIDescriptorAlloc 参数说明 .....	9
3.7. NCIDescriptorFree 参数说明 .....	10
3.8. NCIAAttrSet 参数说明 .....	11
3.9. NCIAAttrSet-attribute 参数说明 .....	11
3.10. NCIAAttrSet-attrtype 参数说明 .....	12
3.11. NCIParamGet 参数说明 .....	13
3.12. NCIAAttrGet 参数说明 .....	14
3.13. NCIAAttrGet-attrtype 参数说明 .....	14
3.14. NCIDescribeAny 参数说明 .....	17
3.15. NCI支持对象描述 .....	17
3.16. NCIServerVersion 参数说明 .....	18
3.17. NCIServerAttach 参数说明 .....	19
3.18. NCIServerDetach 参数说明 .....	19
3.19. NCISessionBegin 参数说明 .....	20
3.20. NCISessionEnd 参数说明 .....	20
3.21. NCILogon 参数说明 .....	21
3.22. NCILogoff 参数说明 .....	22
3.23. NCIErrorGet 参数说明 .....	23
3.24. NCISetPrepare 参数说明 .....	23
3.25. NCISetExecute 参数说明 .....	24
3.26. NCIBindByPos 参数说明 .....	25
3.27. NCIDefineByPos 参数说明 .....	27
3.28. NCISetFetch 参数说明 .....	28
3.29. NCISetFetch2 参数说明 .....	29
3.30. NCIBindByName 参数说明 .....	31
3.31. NCIBindArrayOfStruct 参数说明 .....	32
3.32. NCIDefineArrayOfStruct 参数说明 .....	32
3.33. NCISetSetPieceInfo 参数说明 .....	33
3.34. NCISetGetPieceInfo 参数说明 .....	34
3.35. NCIBindDynamic 参数说明 .....	35
3.36. NCIDefineDynamic 参数说明 .....	37
3.37. NCITransCommit 参数说明 .....	37
3.38. NCITransRollback 参数说明 .....	38
3.39. NCITransStart 参数说明 .....	38
3.40. NCITransDetach 参数说明 .....	39
3.41. NCILobRead 参数说明 .....	40
3.42. LOB数据字符数或者字节数 .....	41
3.43. NCILobWrite 参数说明 .....	42
3.44. LOB函数实际写入的LOB数据长度 .....	43
3.45. NCILobGetLength 参数说明 .....	43
3.46. NCILobCreateTemporary 参数说明 .....	44
3.47. NCILobAssign 参数说明 .....	45
3.48. NCILobFreeTemporary 参数说明 .....	45
3.49. NCILobIsEqual 参数说明 .....	46

3.50.	NCILobEnableBuffering 参数说明	46
3.51.	NCILobDisableBuffering 参数说明	47
3.52.	NCILobFlushBuffer 参数说明	47
3.53.	NCILobAppend 参数说明	48
3.54.	NCILobTrim 参数说明	48
3.55.	NCILobErase 参数说明	49
3.56.	NCILobCopy 参数说明	50
3.57.	NCILobOpen 参数说明	50
3.58.	NCILobClose 参数说明	51
3.59.	NCILobLocatorIsInit 参数说明	52
3.60.	NCILobLocatorAssign 参数说明	52
3.61.	NCIRawPtr 参数说明	53
3.62.	NCIRawSize 参数说明	53
3.63.	NCIRawAllocSize 参数说明	54
3.64.	NCIRawAssignBytes 参数说明	54
3.65.	NCIRawResize 参数说明	55
3.66.	NCIRawAssignRaw 参数说明	55
3.67.	NCIDateTimeConstruct 参数说明	56
3.68.	NCIDateTimeFromText 参数说明	57
3.69.	NCIDateTimeToText 参数说明	58
3.70.	NCIDateTimeGetDate 参数说明	59
3.71.	NCIDateTimeGetTime 参数说明	60
3.72.	NCIIntervalSetYearMonth 参数说明	60
3.73.	NCIIntervalGetYearMonth 参数说明	61
3.74.	NCIIntervalSetDaySecond 参数说明	62
3.75.	NCIIntervalGetDaySecond 参数说明	62
3.76.	NCIIntervalCompare 参数说明	63
3.77.	NCIIntervalAdd 参数说明	64
3.78.	NCIIntervalAssign 参数说明	64
3.79.	NCIIntervalToText 参数说明	65
3.80.	NCIIntervalFromText 参数说明	66
3.81.	NCIIntervalCheck 参数说明	66
3.82.	NCIIntervalDivide 参数说明	67
3.83.	NCIIntervalMultiply 参数说明	67
3.84.	NCIIntervalSubtract 参数说明	68
3.85.	NCINumberFromInt 参数说明	69
3.86.	NCINumberToInt 参数说明	69
3.87.	NCINumberFromText 参数说明	70
3.88.	NCINumberToText 参数说明	71
3.89.	NCINumberFromReal 参数说明	72
3.90.	NCINumberToReal 参数说明	72
3.91.	NCINumberAdd 参数说明	73
3.92.	NCINumberAssign 参数说明	74
3.93.	NCINumberDiv 参数说明	74
3.94.	NCINumberIsInt 参数说明	75
3.95.	NCINumberMod 参数说明	75
3.96.	NCINumberMul 参数说明	76
3.97.	NCINumbersub 参数说明	76
3.98.	NCIDirPathPrepare 参数说明	77
3.99.	NCIDirPathColArrayEntrySet 参数说明	78
3.100.	NCIDirPathColArrayToStream 参数说明	79
3.101.	NCIDirPathLoadStream 参数说明	79
3.102.	NCIDirPathFinish 参数说明	80
3.103.	NCIStringAllocSize 参数说明	80

3.104. NCISStringAssign 参数说明 .....	81
3.105. NCISStringAssignText 参数说明 .....	81
3.106. NCISStringPtr 参数说明 .....	82
3.107. NCISStringResize 参数说明 .....	83
3.108. NCISStringSize 参数说明 .....	83



---

# 前言

## 1. 文档目的

本文档介绍优炫数据库NCI接口的使用。

## 2. 文档对象

- 技术支持工程师
- 维护工程师
- 开发工程师

## 3. 修改记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

表 1. 文档更新记录

数据库版本	发布日期	修改说明
2.1.1.5C	2022-08-25	第一次正式发布。

---

# 第 1 章 概述

NCI（全称Native Call Interface）为访问数据库提供C语言的访问方式，编译和连接一个NCI程序不需要独立的预处理或者预编译步骤。

使用NCI接口接口时，使用-DNCI\_EXPORTT进行指定。

---

# 第 2 章 安装与配置

## 2.1. 安装

1. 确保已经安装uxdb。
2. 获取NCI安装包，请联系优炫工作人员获取。
3. 解压安装包。

```
tar -xf uxdb-nci-12-2.0.tar.gz
```

4. 执行install.sh安装。

执行如下命令。

```
cd uxdb-nci-12-2.0
./install.sh
[uxdb@localhost uxdb-nci-12-2.0]$ ./install.sh
input UXDB_HOME[/opt/uxdbinstall]:
```

```
INSTALL_HOME:/opt/uxdbinstall
installation complete!
```

当提示input UXDB\_HOME[/opt/uxdbinstall]: 时，请输入数据库安装目录后回车，如果直接回车数据库会被安装在默认路径：/opt/uxdbinstall/下。

## 2.2. 配置

- 配置文件路径

数据库安装目录下dbsql/lib/nci/config.ini。

- 配置项

- IsAutoCommit：代表是否自动提交，默认为0。
- IsAllowNullParam：是否允许参数为空，默认为1。
- IsFetchOnExecute：在StmtExecute里面是否执行fetch操作，默认0为不执行。
- uxdbPort：数据库端口，如果配置项错误或者没有配置，默认是5432。
- uxdbName：数据库名称，默认uxdb。

- 参数含义

- IsAutoCommit：当此参数设置为0时，不会对执行语句进行提交，操作只在当前事务可见；如果设置为1时，代表会自动提交，操作对所有事务可见。
- IsAllowNullParam：当此参数设置为0时，代表接口的错误句柄参数和以下表格中的参数不允许为空，为空时会返回错误；如果设置为1时，代表接口的错误句柄参数和以下表格中的参数允许为空，为空时不会返回错误。

表 2.1. 其他参数

接口	参数
NCIDefineByPos	valuep
	rlemp
NCIBindByPos	valuep
NCIIntervalGetDaySecond	dy
	hr
	mm
	ss
	fsec
NCIIntervalGetYearMonth	yr
	mnth
NCIIntervalToText	resultlen
NCIDateTimeGetDate	yr
	mnth
	dy
NCIDateTimeGetTime	hr
	mm
	ss
	fsec

- **IsFetchOnExecute:** 当此参数设置为0时，代表在执行StmtExecute函数处理查询语句后不会往缓冲区存放查询数据。为1时，代表在执行StmtExecute函数处理查询语句后，根据参数items不用执行fetch就会有items条数据被放到缓冲区。
- **uxdbPort:** 数据库端口，nci连接接口如果没有传入数据库端口，就使用配置文件里面的配置参数，如果配置文件里面也没有配置就使用默认端口5432。
- **uxdbName:** 数据库名称，nci连接接口如果没有传入数据库名称，就使用配置文件里面的配置参数，如果配置文件里面也没有配置就使用默认名称uxdb。
- 错误文件

如果读取配置文件参数有错误，在跟config.ini文件同目录的地方会生成错误文件err.log，可以根据错误文件检查配置文件内容哪里有问题。

---

## 第 3 章 接口介绍

### 3.1. 句柄

#### 3.1.1. NCIEnvCreate

- 功能

创建并初始化NCI函数的环境句柄以在其下工作。

- 函数

```
sword NCIEnvCreate(NCIEnv **envhpp,  
ub4 mode,  
CONST dvoid *ctxp,  
CONST dvoid *(*malocfp)(dvoid *ctxp, size_t size),  
CONST dvoid *(*ralocfp)(dvoid *ctxp, dvoid *memptr, size_t newsize),  
CONST void (*mfreefp)(dvoid *ctxp, dvoid *memptr),  
size_t xtramemsz,  
dvoid **usrmempp)
```

- 参数

表 3.1. NCIEnvCreate 参数说明

名称	描述
envhpp (输出)	指向环境句柄的指针。
mode (输入)	指定模式的初始化。  • NCI_DEFAULT: 默认值。
ctxp (输入)	为内存回调例程指定用户定义的上下文。
malocfp (输入)	保留参数。
ralocfp (输入)	保留参数。
mfreefp (输入)	保留参数。
xtramemsz (输入)	保留参数。
usrmempp (输出)	返回一个指针，指向xtramemsz由调用为用户分配的大小的用户内存。

- 注意

此调用使用用户指定的模式为所有NCI调用创建环境，应在任何其他NCI调用之前调用， 常用来代替NCIInitialize()和NCIEnvInit() 的调用。\*envhpp本来就指向一个可用的环境句柄会拒绝分配新的环境句柄并返回NCI\_ERROR。仅可以申请一个环境句柄，重复申请环境句柄会拒绝分配新的环境句柄并返回NCI\_ERROR。

- 示例

```
NCIEnvCreate(&envhpp, NCI_DEFAULT, NULL, NULL, NULL, NULL, 0, NULL);
```

### 3.1.2. NCCHandleAlloc

- 功能

分配和初始化句柄。

- 函数

```
sword NCCHandleAlloc(dvoid *parenth,
dvoid **hndlpp,
CONST ub4 type,
CONST size_t xtrmem_sz,
dvoid **usrmempp)
```

- 参数

表 3.2. NCCHandleAlloc 参数说明

名称	描述
parenth (输入)	环境句柄。
hndlpp (输出)	返回句柄。
type (输入)	<p>从环境句柄上分配的句柄类型。可以分配的句柄类型如下所示。</p> <ul style="list-style-type: none"> <li>• NCI_HTYPE_SERVER: 分配一个NCIServer 结构句柄(连接句柄)，该句柄上存放建立的连接句柄。</li> <li>• NCI_HTYPE_SESSION: 分配一个NCISession 结构句柄(连接信息句柄)，该句柄上存放建立连接所用的登录信息，比如登录的口令和密码。</li> <li>• NCI_HTYPE_SVCCTX: 分配一个NCISvcCtx 结构句柄(上下文句柄)，该句柄上存放的是连接句柄和其他类型句柄的关联信息。当该句柄调用NCITransCommit或NCITransRollback函数提交或回滚时将会影响该句柄所在环境句柄上的所有语句句柄SQL操作。</li> <li>• NCI_HTYPE_STMT: 分配一个NCISstmt结构句柄(语句句柄)，分配该句柄时，环境句柄上必须已经分配了连接句柄并已经建立了连接。可以在一个环境句柄上分配多个语句句柄，每个语句句柄可以进行独立的SQL操作。</li> <li>• NCI_HTYPE_DESCRIBE: 分配一个NCIDescribe结构句柄(描述对象句柄)，该句柄可以用来描述某个数据库中的对象，如表，存储过程等。通过这个句柄，可以得到对象的具体信息，如表结构中的每个列信息，存储过程中的每个参数信息。</li> </ul>

名称	描述
	<ul style="list-style-type: none"><li>• NCI_HTYPE_ERROR: 分配一个NCIError结构句柄(错误描述句柄), 该句柄存放每次 NCI 操作失败的原因, 通过调用NCIErrorGet函数可以获取到这些信息。</li><li>• NCI_HTYPE_DIRPATH_CTX: 分配一个NCIDirPathCtx结构句柄(直接文件操作句柄), 该句柄用于直接文件操作接口。</li><li>• NCI_HTYPE_DIRPATH_COLUMN_ARRAY: 分配一个NCIDirPathColArray结构句柄(直接路径列数组句柄), 该句柄用于直接文件操作接口。</li><li>• NCI_HTYPE_DIRPATH_STREAM: 分配一个NCIDirPathStream结构句柄(数据流描述句柄), 该句柄用于直接文件操作接口。</li></ul>
xtrmem_sz (输入)	保留参数。
usrmempp (输出)	保留参数。

- 注意

此函数调用, 错误时没有可用分析的错误诊断信息。\*hndlpp本来就指向一个可用且类型匹配的句柄会拒绝分配新的句柄并返回错误。例如: 需要分配一个新的语句句柄时, 如果\*hndlpp本来就指向一个语句句柄则会分配失败。

- 示例

```
NCIHandleAlloc((dvoid *)envhpp, (dvoid **)&usrhpp, (ub4)NCI_HTYPE_SESSION, (size_t)0, (dvoid **)0);
```

### 3.1.3. NCIHandleFree

- 功能

显式地释放一个句柄。

- 函数

```
sword NCIHandleFree(void *hndlpp, ub4 type)
```

- 参数

表 3.3. NCIHandleFree 参数说明

名称	描述
hndlpp (输入)	由NCIHandleAlloc分配的句柄。
type (输入)	要释放的句柄类型。

- 返回值

如果释放成功，返回NCI\_SUCCESS，失败返回NCI\_ERROR，如果hdlp指向的句柄无效，返回NCI\_INVALID\_HANDLE。

- 注意

此调用释放与句柄关联的存储空间，对应于type参数中指定的类型。此调用返回NCI\_SUCCESS，NCI\_INVALID\_HANDLE或NCI\_ERROR。所有句柄都可以显式释放。释放环境句柄时，会自动释放环境句柄的子句柄（释放除去define句柄、bind句柄、param句柄、环境句柄以外的其它所有句柄）。

- 示例

```
NCIHandleFree((dvoid *)servhnp, NCI_HTYPE_SERVER);
```

### 3.1.4. NCIInitialize

- 功能

初始化NCI应用环境。

- 函数

```
sword NCIInitialize(ub4 mode,
dvoid *ctxp,
dvoid *(*malocfp)(dvoid *ctxp, size_t size),
dvoid *(*ralocfp)(dvoid *ctxp, dvoid *memptr, size_t newsize),
void (*mfreefp)(dvoid *ctxp, dvoid *memptr))
```

- 参数

表 3.4. NCIInitialize 参数说明

名称	描述
ctxp（输入/输出）	用户定义上下文。
mode（输入）	初始化模式。取值如下： <ul style="list-style-type: none"> <li>• NCI_DEFAULT：缺省模式。</li> </ul>
malocfp（输入）	保留参数。
ralocfp（输入）	保留参数。
mfreefp（输入）	保留参数。
memptr（输入/输出）	内存块的指针。

- 示例

```
NCIInitialize((ub4)NCI_DEFAULT, (dvoid *)0, (dvoid *(*)(dvoid *, size_t))0, (dvoid *(*)(dvoid *,
dvoid *, size_t))0, (void (*)(dvoid *, dvoid *))0);
```

### 3.1.5. NCIEnvInit

- 功能

分配并初始化环境句柄。



- 函数

```
sword NCIEnvInit((NCIEnv **envhp,
ub4 mode,
size_t xtrmem_sz,
dvoid **usrmempp)
```

- 参数

表 3.5. NCIEnvInit 参数说明

名称	描述
mode (输入)	初始化模式，如下所示。 <ul style="list-style-type: none"> <li>• NCI_DEFAULT: 缺省模式。</li> <li>• NCI_ENV_NO_UCB: 在环境初始化时，禁止使用动态回调函数NCIEnvCallback()。</li> </ul>
xtrmem_sz (输入)	指定要在环境持续时间内分配的用户内存大小。
envhp (输出)	指向环境句柄的指针。
usrmempp (输出)	返回一个指向用户内存的指针，该内存大小xtrmemsz由用户在环境持续时间内为用户分配。

- 注意

当调用该函数初始化一个环境句柄以后，必须调用NCIHandleFree来释放这个句柄。当环境句柄分配以后，只允许在这个句柄上分配一个上下文句柄，并只允许建立一个连接。\*envhp本来就指向一个可用的环境句柄会拒绝分配新的环境句柄并返回NCI\_ERROR。仅可以申请一个环境句柄，重复申请环境句柄会拒绝分配新的环境句柄并返回NCI\_ERROR。

- 示例

```
NCIEnvInit(&env, (ub4)NCI_DEFAULT, (size_t)0, (dvoid **)0);
```

### 3.1.6. NCIDescriptorAlloc

- 功能

分配空间以保存描述符或者LOB定位符。

- 函数

```
sword NCIDescriptorAlloc(CONST dvoid *parenth,
dvoid **descpp,
CONST ub4 type,
CONST size_t xtrmem_sz,
dvoid **usrmempp)
```

- 参数

表 3.6. NCIDescriptorAlloc 参数说明

名称	描述
parenth (输入)	环境句柄。
descpp (输出)	返回所需类型的描述符或LOB定位器。
type (输入)	描述符类型，如下所示。 <ul style="list-style-type: none"> <li>• NCI_DTYPE_LOB: 大字段描述。</li> <li>• NCI_DTYPE_NUMBER: 数字描述。</li> <li>• NCI_DTYPE_INTERVAL_YM: 年月类型时间间隔。</li> <li>• NCI_DTYPE_INTERVAL_DS: 天秒类型时间间隔。</li> <li>• NCI_DTYPE_DATE: 日期。</li> <li>• NCI_DTYPE_TIME: 不带时区的时间。</li> <li>• NCI_DTYPE_TIME_TZ: 带时区的时间。</li> <li>• NCI_DTYPE_TIMESTAMP: 不带时区的时间戳。</li> <li>• NCI_DTYPE_TIMESTAMP_TZ: 带时区的时间戳。</li> <li>• NCI_DTYPE_TIMESTAMP_LTZ: 本地时区的时间戳。</li> </ul>
xtrmem_sz (输入)	保留参数。
usrmempp (输入)	保留参数。

- 注意

错误时没有可用的诊断，成功：NCI\_SUCCESS，发生内存不足错误：NCI\_ERROR。\*descpp本来就指向一个可用的描述句柄会拒绝分配新的描述句柄并返回错误。

- 示例

```
NCIDescriptorAlloc(envhp, (dvoid **)lobsrc, (ub4)NCI_DTYPE_LOB, (size_t)0, (dvoid **)0);
```

### 3.1.7. NCIDescriptorFree

- 功能

释放之前分配的描述符。

- 函数

```
sword NCIDescriptorFree(dvoid *descp,
CONST ub4 type);
```

- 参数

表 3.7. NCIDescriptorFree 参数说明

名称	描述
descp (输入)	分配的描述符。
type (输入)	描述符类型，如下所示。 <ul style="list-style-type: none"><li>• NCI_DTYPE_LOB: 大字段描述。</li><li>• NCI_DTYPE_NUMBER: 数字描述。</li><li>• NCI_DTYPE_INTERVAL_YM: 年月类型时间间隔。</li><li>• NCI_DTYPE_INTERVAL_DS: 天秒类型时间间隔。</li><li>• NCI_DTYPE_DATE: 日期。</li><li>• NCI_DTYPE_TIME: 不带时区的时间。</li><li>• NCI_DTYPE_TIME_TZ: 带时区的时间。</li><li>• NCI_DTYPE_TIMESTAMP: 不带时区的时间戳。</li><li>• NCI_DTYPE_TIMESTAMP_TZ: 带时区的时间戳。</li><li>• NCI_DTYPE_TIMESTAMP_LTZ: 本地时区的时间戳。</li></ul>

- 注意

此调用释放与描述符关联的存储。返回NCI\_SUCCESS或NCI\_INVALID\_HANDLE。所有描述符都可以显式释放。

- 示例

```
NCIDescriptorFree((dvoid *) lobp[0], (ub4)NCI_DTYPE_LOB);
```

## 3.2. 属性

### 3.2.1. NCIAAttrSet

- 功能

设置句柄或者描述符的属性。

- 函数

```
sword NCIAAttrSet(dvoid *trghndlp,  
ub4 trghndltyp,  
dvoid *attributep,
```

ub4 size,  
ub4 attrtype,  
NCIError \*errhp)

• 参数

表 3.8. NCIAttrSet 参数说明

名称	描述
trgthndlp (输入)	指向属性被修改的句柄的指针。
trgthndltyp (输入)	trgthndlp参数句柄的类型，类型如下所示。 <ul style="list-style-type: none"> <li>• NCI_HTYPE_SVCCTX: 上下文句柄。</li> <li>• NCI_HTYPE_SESSION: 连接信息句柄。</li> <li>• NCI_HTYPE_DIRPATH_CTX: 直接文件操作句柄。</li> <li>• NCI_DTYPE_PARAM: 参数句柄。</li> </ul>
attributep (输入)	指向属性值的指针。属性值被复制到目标句柄中。如果属性值是指针，则只复制指针，而不复制指针的内容。具体下文所示。
size (输入)	attributep 参数的大小，如果attributep参数指针是一个字符串类型指针，那么size 就是该字符串的实际长度；如果attributep参数是其他类型指针，则忽略该参数。
attrtype (输入)	要设置的句柄属性。不同类型的句柄有这不同的属性，具体下文所示。
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。

attributep指要设置的属性值指针，如下所示是各种属性情况下的数据值类型。

表 3.9. NCIAttrSet-attributep 参数说明

属性值	可选值
NCI_HTYPE_SVCCTX	<ul style="list-style-type: none"> <li>• NCI_ATTR_SERVER: NCIServer结构句柄(连接句柄)。</li> <li>• NCI_ATTR_SESSION: NCISession结构句柄(连接信息句柄)。</li> </ul>
NCI_HTYPE_SESSION	<ul style="list-style-type: none"> <li>• NCI_ATTR_USERNAME: 字符串指针(char*)。</li> <li>• NCI_ATTR_PASSWORD: 字符串指针(char*)。</li> </ul>
NCI_HTYPE_STMT	<ul style="list-style-type: none"> <li>• NCI_ATTR_PREFETCH_ROWS: 无符号整形指针(ub4*)。</li> </ul>
NCI_HTYPE_DIRPATH_CTX	<ul style="list-style-type: none"> <li>• NCI_ATTR_NAME: 字符串指针(char*)。</li> <li>• NCI_ATTR_SCHEMA_NAME: 字符串指针(char*)。</li> </ul>

属性值	可选值
	<ul style="list-style-type: none"> <li>NCI_ATTR_DIRPATH_INPUT: 无符号整形指针 (ub1*)。</li> <li>NCI_ATTR_BUF_SIZE: 无符号整形指针 (ub4*)。</li> </ul>
NCI_DTYPE_PARAM	<ul style="list-style-type: none"> <li>NCI_ATTR_NAME: 字符串指针 (char*)。</li> <li>NCI_ATTR_DATA_TYPE: 无符号整形指针 (ub2*)。</li> <li>NCI_ATTR_PREFETCH_ROWS: 无符号整形指针 (ub4*)。</li> </ul>

attrtype不同类型的句柄有着不同的属性，如下所示。

表 3.10. NCIAAttrSet-attrtype 参数说明

属性值	可选值
NCI_HTYPE_SVCCTX	<ul style="list-style-type: none"> <li>NCI_ATTR_SERVER: 在上下文句柄上附加一个连接句柄。</li> <li>NCI_ATTR_SESSION: 在上下文句柄上附加一个连接信息句柄。</li> </ul>
NCI_HTYPE_SESSION	<ul style="list-style-type: none"> <li>NCI_ATTR_USERNAME: 在连接信息句柄上设置登录的用户名。</li> <li>NCI_ATTR_PASSWORD: 在连接信息句柄上设置登录的口令。</li> </ul>
NCI_HTYPE_STMT	<ul style="list-style-type: none"> <li>NCI_ATTR_PREFETCH_ROWS: 设置预取值，如果设置的预取值大于执行语句结果的总条数，最多只能取出执行语句结果的总条数；如果设置预取值小于执行语句结果的总条数，最多只能取出设置的预取值大小的条数，当再去取得时候会返回NCI_NO_DATA。</li> </ul>
NCI_HTYPE_DIRPATH_CTX	<ul style="list-style-type: none"> <li>NCI_ATTR_NAME: 在直接文件操作句柄上设置操作的表名。</li> <li>NCI_ATTR_SCHEMA_NAME: 在直接文件操作句柄上设置模式名。</li> <li>NCI_ATTR_DIRPATH_INPUT: 在直接文件操作句柄上设置输入类型。</li> <li>NCI_ATTR_BUF_SIZE: 在直接文件操作句柄上设置缓冲区大小。</li> </ul>
NCI_DTYPE_PARAM	<ul style="list-style-type: none"> <li>NCI_ATTR_NAME: 在参数句柄上设置操作的列名。</li> <li>NCI_ATTR_DATA_TYPE: 在参数句柄上设置操作的列类型。</li> </ul>

属性值	可选值
	<ul style="list-style-type: none"> <li>NCI_ATTR_DATA_SIZE: 在参数句柄上设置操作的列取值范围。</li> </ul>

- 注意

NCI\_ATTR\_PREFETCH\_ROWS属性只能在NCISmtPrepare和NCISmtExecute之间进行设置，否则会返回：NCI\_ERROR。

- 示例

```
NCIAttrSet((dvoid *)usrhpp, (ub4)NCI_HTYPE_SESSION, (dvoid *)user, (ub4)strlen(user),
(ub4)NCI_ATTR_USERNAME, errhpp);
```

### 3.2.2. NCIParmGet

- 功能

返回一个描述句柄或语句句柄中指定位置的参数描述符。

- 函数

```
sword NCIParmGet(CONST dvoid *hndl,
ub4 htype,
NCIError *errhp,
dvoid **parmdpp,
ub4 pos)
```

- 参数

表 3.11. NCIParmGet 参数说明

名称	描述
hndl (输入)	语句句柄或描述句柄。NCIParmGet() 函数返回此句柄的参数描述符。
htype (输入)	hndl参数句柄的类型，支持以下两种类型。 <ul style="list-style-type: none"> <li>NCI_HTYPE_STMT: 语句句柄。</li> <li>NCI_DTYPE_PARAM: 参数句柄。</li> </ul>
pos (输入)	语句句柄或描述句柄中的位置编号，为该位置返回一个参数描述符。
errhp (输入)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
parmdpp (输出)	输出的描述符句柄，通过该句柄调用NCIAttrGet 函数就可以得到指定位置上的描述信息。

- 示例

```
NCIParmGet((void *)ph->phNCISmt, NCI_HTYPE_STMT, ph->phNCIErr, (void **)&param, 1);
```

### 3.2.3. NCIAttrGet

- 功能

获取一个句柄的属性。

- 函数

```
sword NCIAttrGet(CONST dvoid *trgthndlp,
ub4 trgthndltyp,
dvoid *attributep,
ub4 *size,
ub4 attrtype,
NCIError *errhp)
```

- 参数

表 3.12. NCIAttrGet 参数说明

名称	描述
trgthndlp (输入)	指向句柄类型的指针。实际句柄可以是语句句柄、会话句柄等，当此调用用于获取编码时，允许用户检查环境或语句句柄。
trgthndltyp (输入)	trgthndlp参数中的句柄类型，类型如下所示。 <ul style="list-style-type: none"> <li>• NCI_DTYPE_PARAM: 参数句柄。</li> <li>• NCI_HTYPE_STMT: 语句句柄。</li> <li>• NCI_HTYPE_DESCRIBE: 描述句柄。</li> <li>• NCI_HTYPE_DIRPATH_CTX: 直接文件操作句柄。</li> </ul>
attributep (输出)	指向属性值存储的指针。
size (输出)	仅兼容，无实际意义。
attrtype (输入)	属性类型。不同类型的句柄有不同属性可以获取。具体如下文所示。
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。

attrtype在不同的输入句柄上的属性取值，如下所示。

表 3.13. NCIAttrGet-attrtype 参数说明

属性值	可选值
NCI_DTYPE_PARAM	<ul style="list-style-type: none"> <li>• NCI_HTYPE_RESULT_COL_ATTR</li> <li>• NCI_ATTR_DATA_SIZE: 列的数据类型大小 (ub2)。</li> </ul>

属性值	可选值
	<ul style="list-style-type: none"><li>• NCI_ATTR_DATA_TYPE: 列的数据类型(ub2)。</li><li>• NCI_ATTR_NAME: 列名(char*)。</li><li>• NCI_ATTR_PRECISION: 列的精度(ub2)。</li><li>• NCI_ATTR_SCALE: 列的刻度(ub1)。</li><li>• NCI_ATTR_IS_NULL: 列是否允许为空(ub1)。</li><li>• NCI_PTYPE_TABLE<ul style="list-style-type: none"><li>• NCI_ATTR_OBJID: 表的ID号(ub4)。</li><li>• NCI_ATTR_NUM_COLS: 表中的列数(ub2)。</li><li>• NCI_ATTR_LIST_COLUMNS: 表中列信息的描述参数句柄(void*)。</li></ul></li><li>• NCI_ATTR_LIST_COLUMNS 同NCI_HTYPE_RESULT_COL_ATTR属性</li><li>• NCI_PTYPE_PROC<ul style="list-style-type: none"><li>• NCI_ATTR_DATA_SIZE: 参数的数据类型大小(ub2)。</li><li>• NCI_ATTR_DATA_TYPE: 参数的数据类型(ub2)。</li><li>• NCI_ATTR_NAME: 参数名(char*)。</li><li>• NCI_ATTR_PRECISION: 参数的精度(ub1)。</li><li>• NCI_ATTR_SCALE: 参数的刻度(ub1)。</li><li>• NCI_ATTR_NUM_PARAMS: 参数的个数(ub2)。</li><li>• NCI_ATTR_LIST_ARGUMENTS: 参数的描述参数句柄(void*)。</li></ul></li><li>• NCI_PTYPE_FUNC 同NCI_PTYPE_PROC</li><li>• NCI_PTYPE_VIEW<ul style="list-style-type: none"><li>• NCI_ATTR_OBJID: 视图的ID号(ub4)。</li></ul></li></ul>



属性值	可选值
	<ul style="list-style-type: none"> <li>• NCI_ATTR_NUM_COLS: 视图中的列数 (ub2)。</li> <li>• NCI_ATTR_LIST_COLUMNS: 视图中列信息的描述参数句柄(void*)。</li> <li>• NCI_PTYPE_TYPE</li> <li>• NCI_ATTR_OBJID: 类型的ID号 (ub4)。</li> <li>• NCI_PTYPE_SCHEMA</li> <li>• NCI_ATTR_OBJID: 模式的ID号 (ub4)。</li> <li>• NCI_PTYPE_DATABASE</li> <li>• NCI_ATTR_OBJID: 数据库的ID号 (ub4)。</li> </ul>
NCI_HTYPE_STMT	<ul style="list-style-type: none"> <li>• NCI_ATTR_NUM_COLS: 结果集中列的个数 (ub2)。</li> <li>• NCI_ATTR_ROW_COUNT: 已经返回记录的总行数 (ub4)。</li> </ul>
NCI_HTYPE_DESCRIBE	<ul style="list-style-type: none"> <li>• NCI_ATTR_PARAM: 获取参数句柄。</li> </ul>
NCI_HTYPE_DIRPATH_CTX	<ul style="list-style-type: none"> <li>• NCI_ATTR_LIST_COLUMNS: 在直接文件操作句柄上获取参数句柄(NCIParam*)。</li> </ul>

- 注意

此调用用于获取句柄的特定属性。NCI\_DTYPE\_PARAM用于进行隐式和显式描述。参数描述符也用于直接路径加载。对于隐式描述，参数描述符具有每个选择列表的列描述。对于显式描述，参数描述符具有描述的每个模式对象的描述信息。

- 示例

```
NCIAttrGet(param, NCI_DTYPE_PARAM, &colname, &size, NCI_ATTR_NAME, ph->phNCIErr);
```

### 3.2.4. NCIDescribeAny

- 功能

连续写入内容到一个大字段存储描述符中。

- 函数

```
sword NCIDescribeAny (NCISvcCtx *svchp,
NCIError *errhp,
dvoid *objptr,
ub4 objptr_len,
ub1objptr_typ,
ub1 info_level,
ub1 objtyp,
NCIDescribe *dschp);
```

- 参数

表 3.14. NCIDescribeAny 参数说明

名称	描述
svchp (输入)	上下文句柄指针。
errhp (输入/输出)	错误信息句柄。
objptr (输入)	要被描述的对象指针，目前只支持字符串类型指针。
objnm_len (输入)	objptr参数中字符串的长度。
objptr_typ (输入)	objptr指针类型，目前只支持 NCI_OTYPE_NAME - 对象名称类型指针。
info_level (输入)	保留参数。
objtyp (输入)	objtyp参数所指的对象类型，可以为下面几种对象。 <ul style="list-style-type: none"><li>• NCI_PTYPE_TABLE: 表</li><li>• NCI_PTYPE_PROC: 过程</li><li>• NCI_PTYPE_FUNC: 函数</li><li>• NCI_PTYPE_VIEW: 视图</li><li>• NCI_PTYPE_TYPE: 类型</li><li>• NCI_PTYPE_SCHEMA: 模式</li><li>• NCI_PTYPE_DATABASE: 数据库</li></ul>
dschp (输入/输出)	一个描述符句柄，该句柄可以通过使用 NCI_HTYPE_DESCRIBE。作为参数调用 NCIHandleAlloc分配得到。

- 返回值

成功返回NCI\_SUCCESS，失败返回NCI\_ERROR。

- 注意

表 3.15. NCI支持对象描述

句柄	描述
NCI_PTYPE_TABLE	表
NCI_PTYPE_PROC	存储过程
NCI_PTYPE_FUNC	函数
NCI_PTYPE_VIEW	视图
NCI_PTYPE_TYPE	类型
NCI_PTYPE_SCHEMA	模式
NCI_PTYPE_DATABASE	数据库

- 示例

```
NCIDescribeAny(ph->phNCISvctx, ph->phNCIErr, (dvoid *)objptr, objp_len, NCI_OTYPE_NAME,
0, NCI_PTYPE_TABLE, dschp);
```

### 3.2.5. NCIServerVersion

- 功能

获取版本信息。

- 函数

```
sword NCIServerVersion(dvoid *hndlp,
NCIError *errhp,
text *bufp,
ub4 bufisz,
ub1 hndltype)
```

- 参数

表 3.16. NCIServerVersion 参数说明

名称	描述
hndlp (输入)	服务上下文句柄或服务器上下文句柄。
bufisz (输入)	缓冲区的长度 (以字节数为单位)。
hndltype (输入)	传递给函数的句柄类型。
errhp (输入)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
bufp (输入)	返回版本信息的缓冲区。

- 示例

```
NCIServerVersion(ph->phNCISvctx, ph->phNCIErr, buff, BUFF_SIZE, NCI_HTYPE_SVCCTX);
```

## 3.3. 连接

### 3.3.1. NCIServerAttach

- 功能

将一个数据库服务挂载到一个指定的连接句柄上。

- 函数

```
sword NCIServerAttach(NCIServer *srvhp,
NCIError *errhp,
CONST NCIText *dblink,
sb4 dblink_len,
ub4 mode)
```

- 参数

表 3.17. NCIServerAttach 参数说明

名称	描述
srvhp (输入)	一个未初始化的服务器句柄，由此处初始化。
errhp (输入)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
dblink (输入)	指定要使用的数据库服务器。此参数指向指定连接字符串或服务点的字符串。如果连接字符串是NULL，返回NCI_ERROR。
dblink_len (输入)	dblink的长度，对于有效的连接字符串名称或别名，dblink_len必须非零。它的值以字节数为单位。
mode (输入)	指定操作模式。仅支持NCI_DEFAULT。

- 示例

```
NCIServerAttach(srvhp, errhp, (text *)dbname, strlen(dbname), NCI_DEFAULT);
```

### 3.3.2. NCIServerDetach

- 功能

删除NCI操作对一个数据源的访问。

- 函数

```
sword NCIServerDetach(NCIServer *srvhp,
NCIError *errhp,
ub4 mode)
```

- 参数

表 3.18. NCIServerDetach 参数说明

名称	描述
srvhp (输入)	初始化服务器上下文的句柄，它被重置为未初始化状态。
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
mode (输入)	附加模式。仅支持NCI_DEFAULT。

- 示例

```
NCIServerDetach(srvhpp, errhpp, NCI_DEFAULT);
```

### 3.3.3. NCISessionBegin

- 功能

创建并开始一个用户会话。

- 函数

```

sword NCISessionBegin(NCISvcCtx *svchp,
    NCLError *errhp,
    NCISession *usrhp,
    ub4 credt,
    ub4 mode)

```

- 参数

表 3.19. NCISessionBegin 参数说明

名称	描述
svchp (输入)	服务上下文的句柄，必须设置且需有效。
credt (输入)	登录的方式，NCI_CRED_RDBMS：通过用户名和口令与数据库服务建立起连接。
mode (输入)	指定操作模式，有效模式如下所示。 <ul style="list-style-type: none"> <li>• NCI_DEFAULT：在此模式下，返回的用户会话上下文只能使用svchp指定的服务器上下文进行设置。</li> </ul>
errhp (输入)	出现错误时可以传递给NCLErrorGet() 诊断信息的错误句柄。
usrhp (输入)	用户会话上下文的句柄，由此调用初始化。

- 示例

```
NCISessionBegin(svchpp, errhpp, usrhpp, NCI_CRED_RDBMS, (ub4)NCI_DEFAULT);
```

### 3.3.4. NCISessionEnd

- 功能

终止NCISessionBegin() 函数所创建的用户会话。

- 函数

```

sword NCISessionEnd(NCISvcCtx *svchp,
    NCLError *errhp,
    NCISession *usrhp,
    ub4 mode)

```

- 参数

表 3.20. NCISessionEnd 参数说明

名称	描述
usrhp (输入)	取消对该用户的身份验证。如果此参数作为NULL传递，则返回NCI_INVALID_HANDLE。
mode (输入)	连接模式，取值如下： <ul style="list-style-type: none"> <li>• NCI_DEFAULT：缺省模式。</li> </ul>

名称	描述
svchp (输入)	服务上下文句柄。必须有一个与关联的有效服务器句柄和用户会话句柄。
errhp (输入)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。

- 示例

```
NCISessionEnd(svchp, errhp, authp, (ub4) 0);
```

### 3.3.5. NCILogon

- 功能

根据用户名和密码，登录到一个指定的数据库服务上，并初始化相关的上下文句柄。当连接建立以后，需要通过调用 NCILogoff 来断开连接。

- 函数

```
sword NCILogon(NCIEnv *envhp,
NCIError *errhp,
NCISvcCtx **svchp,
const NCIText *username,
ub4 uname_len,
const NCIText *password,
ub4 passwd_len,
const NCIText *dbname,
ub4 dbname_len)
```

- 参数

表 3.21. NCILogon 参数说明

名称	描述
envhp (输入)	NCI环境句柄。
username (输入)	用户名。
uname_len (输入)	用户名长度。
password (输入)	用户密码。
passwd_len (输入)	密码长度。
dbname (输入)	要连接的数据库名称。其形式为ip:port/databasename或ip:port或ip。
dbname_len (输入)	数据库名称长度。
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
svchp (输入/输出)	服务上下文指针。如果指定的上下文句柄非空，那么在登录以后，会自动设置相关的连接句柄到该句柄上。如果指定的上下文句柄为空，那么 NCI 会自动分配一个上下文句柄输出，该上下文句柄不需调用 NCIHandleFree

名称	描述
	释放；当应用调用 NCILogoff 以后，NCI会自动释放这个句柄。

- 注意

此函数用于为应用程序创建一个简单的登录会话。

- 示例

```
NCILogon(env, error, &hdbc, username, sizeof(dbname), pwd, sizeof(pwd), dbname, sizeof(dbname));
```

### 3.3.6. NCILogoff

- 功能

断开通过NCILogon与服务器建立的连接。

- 函数

```
sword NCILogoff(NCISvcCtx *svchp,  
NCIError *errhp)
```

- 参数

表 3.22. NCILogoff 参数说明

名称	描述
svchp（输入）	服务上下文句柄。
errhp（输入/输出）	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。

- 注意

在默认情况下，此函数关闭会话或连接。

- 示例

```
NCILogoff(hdbc, 0);
```

## 3.4. 错误

### 3.4.1. NCIErrorGet

- 功能

获取NCI操作失败产生的错误信息。

- 函数

```
sword NCIErrorGet(dvoid *hndlp,  
ub4 recordno,  
NCIText *sqlstate,  
sb4 *errcodep,
```

```
NCIText *bufp,
ub4 bufsiz,
ub4 type)
```

- 参数

表 3.23. NCLErrorGet 参数说明

名称	描述
hndlp (输入)	错误信息句柄或环境信息句柄。
recordno (输入)	指示应用程序从中寻求信息的状态记录。从1开始。
sqlstate (输出)	SQL语句执行状态。
errcodep (输出)	返回的错误代码。
bufp (输出)	返回的错误消息文本。
bufsiz (输入)	为错误消息提供的缓冲区的大小，以字节数为单位。
type (输入)	句柄的类型 (NCI_HTYPE_ERROR或NCI_HTYPE_ENV)。

- 示例

```
NCLErrorGet(ph->phNCIErr, 1, sqlstate, &errcodep, (NCIText *)buff,
BUFF_SIZE,NCI_HTYPE_ERROR);
```

## 3.5. 语句

### 3.5.1. NCISstmtPrepare

- 功能

执行SQL语句而做准备。

- 函数

```
sword NCISstmtPrepare(NCISstmt *stmtp,
NCLError *errhp,
NCIText *stmt,
ub4 stmt_len,
ub4 language,
ub4 mode)
```

- 参数

表 3.24. NCISstmtPrepare 参数说明

名称	描述
stmtp (输入)	与要执行的语句关联的语句句柄。
errhp (输入)	出现错误时可以传递给NCLErrorGet() 诊断信息的错误句柄。



名称	描述
stmt（输入）	要执行的SQL语句。若准备多条SQL语句，则语句之间用分号隔开。
stmt_len（输入）	语句的长度，以字符或字节数表示，具体取决于编码，不得为0。
language（输入）	保留参数。
mode（输入）	准备模式，取值如下： <ul style="list-style-type: none"> <li>NCI_DEFAULT：缺省模式。</li> </ul>

- 示例

```
NCISmtPrepare(stmthpp, errhpp, (text *)sql1, strlen(sql1), (ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT);
```

### 3.5.2. NCISmtExecute

- 功能

执行通过NCISmtPrepare准备后的语句。

- 函数

```
sword NCISmtExecute(NCISvcCtx *svchp,
NCISmt *stmtp,
NCIError *errhp,
ub4 iters,
ub4 rowoff,
const NCISnapshot *snap_in,
NCISnapshot *snap_out,
ub4 mode)
```

- 参数

表 3.25. NCISmtExecute 参数说明

名称	描述
svchp（输入）	服务上下文句柄。
stmtp（输入）	一个语句句柄。它定义了要在服务器上执行的语句和相关数据。
errhp（输入）	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
iters（输入）	执行后影响的行数。对非查询语句，该参数表明一次性向服务器发送的数据行数，这个时候，该值必须大于等于1；对于查询语句，如果配置文件的IsFetchOnExecute值为0，此参数被忽略，如果IsFetchOnExecute值为1，将会有iters条数据被放入缓冲区。
rowoff（输入）	行集偏移。对于查询语句，该参数将被忽略；对于非查询语句，该参数表明向服务器发送多行数据时的起始偏移行，从0开始计算。

名称	描述
snap_in (输入)	保留参数。
snap_out (输入)	保留参数。
mode (输入)	<p>执行的模式，如下所示。</p> <ul style="list-style-type: none"> <li>• NCI_DEFAULT: 缺省模式。</li> <li>• NCI_COMMIT_ON_SUCCESS: 自动提交模式。当SQL执行以后，会自动提交执行的SQL。</li> <li>• NCI_DESCRIBE_ONLY: 描述模式。通过该模式执行准备好的语句，并不是真正的执行语句，而是返回语句中的结果集描述，应用程序可以通过调用NCIAttrGet 函数来获取这些信息。</li> <li>• NCI_EXACT_FETCH: 提取模式。当执行完以后，可以从结果集中提取数据。</li> </ul>

- 示例

```
NCISmtfExecute(svchpp, stmthpp, errhpp, (ub4)0, (ub4)0, NULL, NULL, NCI_DEFAULT);
```

### 3.5.3. NCIBindByPos

- 功能

创建程序变量和一个SQL语句中的占位符之间的结合。

- 函数

```
sword NCIBindByPos(NCISmt *stmtp,
NCIBind **bindp,
NCIError *errhp,
ub4 position,
dvoid *valuep,
sb4 value_sz,
ub2 dty,
dvoid *indp,
ub2 *alenp,
ub2 *rcodep,
ub4 maxarr_len,
ub4 *curelep,
ub4 mode)
```

- 参数

表 3.26. NCIBindByPos 参数说明

名称	描述
stmtp (输入/输出)	正在处理的SQL语句的句柄。

名称	描述
bindp (输入/输出)	输出的绑定信息句柄。可以通过该句柄调用 NCIBindArrayOfStruct 函数来设定该参数每行的间隔长度。
errhp (输入/输出)	出现错误时可以传递给 NCLErrorGet() 诊断信息的错误句柄。
position (输入)	如果 NCIBindByPos() 被调用, 则占位符属性由位置指定。位置是从1开始。
valuep (输入/输出)	dtypes 参数中指定类型的数据值或数据值数组的地址。可以指定数据值数组为 SQL 多行操作提供数据。
value_sz (输入)	valuep 的最大可能大小 (以字节为单位)。
dtype (输入)	被绑定值的数据类型。
indp (输入)	保留参数。
alenp (输入)	保留参数。
rcodep (输入)	保留参数。
maxarr_len (输入)	保留参数。
curelep (输入)	保留参数。
mode (输入)	绑定模式, 如下所示。 <ul style="list-style-type: none"> <li>• NCI_DEFAULT: 缺省模式。</li> <li>• NCI_DATA_AT_EXEC: 数据在执行时获取, 此时指定的 valuep 无效 (需要搭配 NCISetStmtSetPieceInfo 或者 NCIBindDynamic 使用, 由其指定动态绑定地址或分片绑定地址, 如未指定, NCISetStmtExecute 调用会返回 NCI_NEED_DATA)。</li> </ul>

- 示例

```
NCIBindByPos(stmthp, &bndhp[0], errhp, (ub4) 1, (dvoid *) &in1[0], (sb4) sizeof(in1[0]),
SQLT_INT, (dvoid *) 0, (ub2 *) 0, (ub2 *) 0, (ub4) 0, (ub4 *) 0, (ub4) NCI_DEFAULT);
```

### 3.5.4. NCIDefineByPos

- 功能

将选择列表中的项目与类型和输出数据缓冲区相关联。

- 函数

```
sword NCIDefineByPos(NCISet *stmtp,
NCIDefine **defnp,
NCIError *errhp,
ub4 position,
dvoid *valuep,
sb4 value_sz,
ub2 dtype,
dvoid *indp,
```

```
ub2 *rlenp,
ub2 *rcodep,
ub4 mode)
```

- 参数

表 3.27. NCIDefineByPos 参数说明

名称	描述
stmtp (输入)	请求的SQL查询操作的句柄。
defnp (输入/输出)	绑定结构输出指针。
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
position (输入)	此值在选择列表中的位置。位置从1开始，从左到右编号。
valuep (输入/输出)	指向dty参数中指定类型的缓冲区或缓冲区数组的指针。
value_sz (输入)	每个valuep缓冲区的大小（以字节为单位）。
dty (输入)	数据类型。
indp (输入)	指示符缓冲区，如果结果集中存在NULL值，那么NCI会回填该缓冲区，把对应的位置置为-1，其他情况下置 0；如果在绑定的时候，未设置该指示符缓冲区，但是获取的结果中又存在 NULL 值，结果集提取时会返回 NCI_ERROR的错误
rlenp (输入/输出)	返回值长度指示缓冲区，如果返回值未发生截断，那么该缓冲区中的值就是实际的返回值长度；如果返回值在获取的时候发生了截断，那么该缓冲区的值就是未截断以前的长度；如果返回值是空值，该缓冲区的值将被置为 0。
rcodep (输入)	保留参数。
mode (输入)	绑定模式，如下所示。 <ul style="list-style-type: none"> <li>NCI_DEFAULT：缺省模式。</li> <li>NCI_DYNAMIC_FETCH：动态提取，此时指定的valuep无效(需要搭配 NCISmtGetPieceInfo、NCISmtSetPieceInfo 或者NCIBindDynamic使用，由其指定动态绑定地址或分片绑定地址，如未指定，NCISmtFetch、NCISmtFetch2调用会返回NCI_NEED_DATA)。</li> </ul>

- 示例

```
NCIDefineByPos(stmthpp, &bhp2, errhpp, 2, (dvoid *)&val, sizeof(val), SQLT_INT,
NULL,&datalen, NULL, NCI_DEFAULT);
```

### 3.5.5. NCISmtFetch

- 功能

获取查询的结果。

- 函数

```
sword NCISmtFetch(NCISmt *stmtp,  
NCIError *errhp,  
ub4 nrows,  
ub2 orientation,  
ub4 mode)
```

- 参数

表 3.28. NCISmtFetch 参数说明

名称	描述
stmtp (输入)	请求的SQL查询操作的句柄。
errhp (输入)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
nrows (输入)	要从当前位置获取的行数。
orientation (输入)	行集提取的方式，可以有以下几种方式。 <ul style="list-style-type: none"><li>• NCI_FETCH_NEXT: 从当前游标位置向下进行提取操作。</li><li>• NCI_FETCH_FIRST: 从结果集的第一行开始向下进行提取操作。</li><li>• NCI_FETCH_LAST: 从结果集的最后一行开始向下提取操作。</li><li>• NCI_FETCH_PRIOR: 从结果集的当前游标位置向上进行提取操作。</li></ul>
mode (输入)	提取模式，取值为NCI_DEFAULT: 缺省模式。

- 注意

如果语句句柄设置了预取值属性时，并且预取值小于执行语句结果集条数，那么最多只能取到预取值大小的条数，当获取的条数大于预取值时会返回NCI\_NO\_DATA。当需要一次获取多行数据值时，取到的实际条数如果小于nrows会返回NCI\_NO\_DATA。

- 示例

```
NCISmtFetch(stmthpp, errhpp, 1, NCI_FETCH_NEXT, NCI_DEFAULT);
```

### 3.5.6. NCISmtFetch2

- 功能

获取SQL语句生成的(可滚动)结果集中的一行。可用该函数替代 NCISmtFetch() 函数。

- 函数

```
sword NCISmtFetch2(NCISmt *stmtp,
NCIError *errhp,
ub4 nrows,
ub2 orientation,
sb4 fetchOffset,
ub4 mode)
```

- 参数

表 3. 29. NCISmtFetch2 参数说明

名称	描述
stmtp (输入)	结果集语句句柄。
errhp (输入)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
nrows (输入)	要从当前位置获取的行数。
orientation (输入)	行集提取的方式，可以有以下几种方式。 <ul style="list-style-type: none"> <li>• NCI_DEFAULT：与 NCI_FETCH_NEXT 作用相同。</li> <li>• NCI_FETCH_CURRENT：提取当前行。</li> <li>• NCI_FETCH_NEXT：提取当前游标指针位置后的数据行。默认为该方式(与 NCI_DEFAULT 相同)，用于不可滚动的语句句柄。</li> <li>• NCI_FETCH_FIRST：取结果集的第一行。</li> <li>• NCI_FETCH_LAST：取结果集的最后一行。</li> <li>• NCI_FETCH_PRIOR：提取结果集中当前游标指针位置之前的数据行。</li> <li>• NCI_FETCH_ABSOLUTE：获取结果集中的绝对行号(由fetchOffset参数指定)。</li> <li>• NCI_FETCH_RELATIVE：获取结果集中的相对行号(由fetchOffset参数指定)。</li> </ul>
fetchOffset(输入)	和orientation参数一起来改变当前行的位置。
mode (输入)	提取模式，取值为NCI_DEFAULT：缺省模式。

- 注意

如果语句句柄设置了预取值属性时，并且预取值小于执行语句结果集条数，那么最多只能取到预取值大小的条数，当获取的条数大于预取值时会返回NCI\_NO\_DATA。当需要一次获取多行数据值时，取到的实际条数如果小于nrows会返回NCI\_NO\_DATA。

该调用与 NCISmtFetch() 调用类似，只是多了参数 fetchOffset。它可以用于所有的语句句柄，不论是否可滚动。对于一个不可滚动的语句句柄，orientation 参数只能取 NCI\_FETCH\_NEXT，忽略偏移量fetchOffset。

orientation 设置为 NCI\_FETCH\_RELATIVE 时偏移量fetchOffset等效值如下所示。

1. NCI\_FETCH\_CURRENT: 偏移量的值为0。
2. NCI\_FETCH\_NEXT: 偏移量的值为1。
3. NCI\_FETCH\_PRIOR: 偏移量的值为-1。
4. NCI\_ATTR\_ROW\_COUNT的值为获取的最大绝对行号。

所有其他orientation模式，除了NCI\_FETCH\_ABSOLUTE与NCI\_FETCH\_RELATIVE 之外都忽略fetchOffset值。

该调用也可以通过使用NCI\_FETCH\_LAST，然后以 NCI\_ATTR\_CURRENT\_POSITION 方式调用NCIAttrGet() 函数来获取结果集中的行数。

该函数返回值与NCISmtFetch相同。除了返回代码NCI\_NO\_DATA，每一次对于可滚动语句句柄的提取(或执行)操作完成后，并不是应用程序需要的所有行都被返回，此时会发送错误返回代码。

可滚动的语句句柄需要被显式地撤销(即提取0行)或释放，从而才能释放当前滚动游标所占用的服务器资源。不可滚动的语句句柄在收到错误代码后会自动释放。

使用NCI\_ATTR\_ROWS\_FETCHED可获取最后一次成功提取到用户缓冲区中的行数。

- 示例

```
NCISmtFetch2(stmthpp, errhpp, 1, NCI_FETCH_NEXT, 1, NCI_DEFAULT);
```

### 3.5.7. NCIBindByName

- 功能

创建程序变量和一个SQL语句中的占位符之间的结合。

- 函数

```
sword NCIBindByName(NCISmt *stmp,
NCIBind **bindpp,
NCIError *errhp,
CONST NCIText *placeholder,
sb4 placeh_len,
dvoid *valuep,
sb4 value_sz,
ub2 dty,
dvoid *indp,
ub2 *alenp,
ub2 *rcodep,
ub4 maxarr_len,
ub4 *curelep,
ub4 mode)
```

- 参数

表 3.30. NCIBindByName 参数说明

名称	描述
stmt (输入/输出)	绑定影响的语句句柄。
bindp (输入/输出)	输出的绑定信息句柄。可以通过该句柄调用 NCIBindArrayOfStruct 函数来设定该参数每行的间隔长度。
errhp (输入/输出)	错误信息句柄，是传给函数 NCIErrGet() 处理获取诊断信息的句柄。
placeholder (输入)	绑定的参数名称。
placeh_len (输入)	参数名称的长度。
valuep (输入/输出)	参数值缓冲区指针。
value_sz (输入)	参数类型单个值的大小。
dt (输入)	参数的数据类型。
indp (输入/输出)	指示器变量或数组的指针。
alenp (输入)	保留参数。
rcodep (输入)	保留参数。
maxarr_len (输入)	保留参数。
curelep (输入)	保留参数。
mode (输入)	绑定模式，取值如下所示。 <ul style="list-style-type: none"> <li>• NCI_DEFAULT: 缺省模式。</li> <li>• NCI_DATA_AT_EXEC: 数据在执行时获取，此时指定的 valuep 无效(需要搭配 NCISetPieceInfo 或者 NCIBindDynamic 使用，由其指定动态绑定地址或分片绑定地址，如未指定，NCISetExecute 调用会返回 NCI_NEED_DATA)。</li> </ul>

- 注意

当不再需要分配的缓冲区时，它们应该由客户端释放。

- 示例

```
NCIBindByName(ph->phNCISmt, &bhp[1], ph->phNCIErr, ":Vdata2", strlen(":Vdata2"), raw2,
sizeof(raw2), SQLT_BIN, 0,0,0,0,0, NCI_DEFAULT);
```

### 3.5.8. NCIBindArrayOfStruct

- 功能

指定参数绑定时，每个参数项中值的间隔大小，以字节计算(以数组方式进行参数绑定)。

- 函数

```
sword NCIBindArrayOfStruct(NCIBind *bindp,
NCIErr *errhp,
```



ub4 pvskip,  
ub4 indskip,  
ub4 alskip,  
ub4 rcskip)

- 参数

表 3.31. NCIBindArrayOfStruct 参数说明

名称	描述
bindp (输入)	参数绑定结构指针，它来自NCIBindByName或NCIBindByPos的参数绑定函数输出。
errhp (输入)	错误信息句柄，是传给函数NCIErrorGet()处理获取诊断信息的句柄。
pvskip (输入)	参数值下一个值的间隔长度。在多行绑定时，可以通过指定该参数来找到下一行值所在的位置。
indskip (输入)	参数值下一个指示器或结构的间隔长度。
alskip (输入)	参数值下一个实际长度的间隔长度。
rcskip (输入)	参数值下一个列级返回码值的间隔值。

- 示例

```
NCIBindArrayOfStruct(bndhp[0], errhp, sl, indsk[0], rlsk[0], rcsk[0]);
```

### 3.5.9. NCIDefineArrayOfStruct

- 功能

指定行集中每一列中每行值存贮位置间隔的大小，以字节计算(以数组方式进行参数绑定)。

- 函数

```
sword NCIDefineAny ((NCIDefine* defnp,  
NCIError* errhp,  
ub4 pvskip,  
ub4 indskip,  
ub4 alskip,  
ub4 rcskip);
```

- 参数

表 3.32. NCIDefineArrayOfStruct 参数说明

名称	描述
defnp (输入)	参数定义结构指针，它来自NCIDefineByName或NCIDefineByPos的参数绑定函数输出。
errhp (输入)	错误信息句柄，是传给函数NCIErrorGet()处理获取诊断信息的句柄。
pvskip (输入)	参数值下一个值的间隔长度。在多行输出绑定时，可以通过指定该参数来找到下一行值所在的位置。

名称	描述
indskip (输入)	参数值下一个指示器或结构的间隔长度。
alskip (输入)	参数值下一个实际长度的间隔长度。
rcskip (输入)	保留参数。

- 示例

```
NCIDefineArrayOfStruct(defnp [0], errhp, sl, indsk[0], rlsk[0], rcsk[0]);
```

### 3.5.10. NCISmtSetPieceInfo

- 功能

设置分片信息。

- 函数

```
sword NCISmtSetPieceInfo ( dvoid *hndlp,
    ub4 type,
    NCIError *errhp,
    CONST dvoid *bufp,
    ub4 *alenp,
    ub1 piece,
    CONST dvoid *indp,
    ub2 *rcodep );
```

- 参数

表 3.33. NCISmtSetPieceInfo 参数说明

名称	描述
hndlpp(输入/输出)	绑定/定义句柄。
type(输入)	句柄的类型。
errhp(输出)	错误句柄，调用失败时，可通过 NCIErrorGet() 可获取错误诊断信息。
bufp(输入/输出)	当 bufp 作为输入型绑定变量时，它是指向一个存储包含数值或分段信息的存储空间的指针；作为 输出型绑定或定义变量时，它指向一个获取分段信息或数值的存储空间。
alenp(输入/输出)	分段信息或值的字节长度。
piecep(输入)	分段参数，仅用于输入绑定变量，取值如下所示。 <ul style="list-style-type: none"> <li>• NCI_ONE_PIECE</li> <li>• NCI_FIRST_PIECE</li> <li>• NCI_NEXT_PIECE</li> <li>• NCI_LAST_PIECE</li> </ul>
indp(输入/输出)	仅做兼容。

名称	描述
rcodep(输入/输出)	仅做兼容。

- 注意

1. 仅允许1行数据做分片。
2. 如果还有分片数据执行NCISmtExecute、NCISmtFetch、NCISmtFetch2会返回NCI\_NEED\_DATA。

- 示例

```
NCISmtSetPieceInfo((dvoid *)defnp[1], (ub4)hdltype, error, (dvoid *)bufout, &alenp, piece, (dvoid *)&indptr, &rcode);
```

### 3.5.11. NCISmtGetPieceInfo

- 功能

获取分片信息。

- 函数

```
sword NCISmtGetPieceInfo ( CONST NCISmt *stmtp,
    NCLError *errhp,
    dvoid **hdlpp,
    ub4 *typep,
    ub1 *in_outp,
    ub4 *iterp,
    ub4 *idxp,
    ub1 *piecep );
```

- 参数

表 3.34. NCISmtGetPieceInfo 参数说明

名称	描述
stmthp(输入)	语句句柄。
errhp(输出)	错误句柄，调用失败时，可通过NCLErrorGet()可获取错误诊断信息。
hdlpp(输出)	返回一个指向绑定或定义句柄的指针，该句柄提供或需要当前的运行数据。
typep(输出)	需要分片输出返回NCI_HTYPE_DEFINE。
in_outp(输出)	需要分片输出返回 NCI_PARAM_OUT。
iterp(输出)	仅做兼容。
idxp(输出)	仅做兼容。
piecep(输出)	返回以下值中的一个： NCI_FIRST_PIECE, NCI_NEXT_PIECE。

- 注意

仅允许1行数据做分片。

- 示例

```
NCIStmtGetPieceInfo(stmt, error, (void **)&defnp[1], &hdltype, &in_out, &iter, &idx, &piece);
```

### 3.5.12. NCIBindDynamic

- 功能

通过回调函数动态数据绑定。

- 函数

```
sword NCIBindDynamic(NCIBind *bindp,
NCIError *errhp,
dvoid *ictxp,
NCICallbackInBind (icbfp)(/*_
dvoid *ictxp,
NCIBind *bindp,
ub4 iter,
ub4 index,
dvoid **bufpp,
ub4 *alenp,
ub1 *piecep,
dvoid **indpp */),
dvoid *octxp,
NCICallbackOutBind(ocbfp)(/*_
dvoid *octxp,
NCIBind *bindp,
ub4 iter,
ub4 index,
dvoid **bufpp,
ub4 *alenpp,
ub1 *piecep,
dvoid **indpp,
ub2 **rcodepp _ */);
```

- 参数

表 3.35. NCIBindDynamic 参数说明

名称	描述
bindp (IN/OUT)	绑定信息句柄（通过调用NCIBindByName或者NCIBindByPos获得）。
errhp (IN/OUT)	错误句柄。
ictxp (IN/OUT)	绑定输入回调函数所需的上下文指针。
icbfp (IN)	指向IN的回调函数。
ictxp (IN)	此回调函数的上下文指针。
bindp (IN)	传入的绑定句柄。
iter (IN)	仅做兼容。
index (IN)	仅做兼容。
bufpp (OUT)	指向缓冲区指针。

名称	描述
alenp (OUT)	用于读取绑定值后填充大小的指针。
piecep (OUT)	仅做兼容。
indpp (OUT)	绑定列的类型指针。
Octxp (IN)	回调函数的上下文指针。
ocbfp (IN)	绑定输出的回调函数。
octxp (IN/OUT)	绑定输出回调函数所需的上下文指针。
bindp (IN)	绑定句柄标识符。
iter (IN)	仅做兼容。
index (IN)	仅做兼容。
bufpp (OUT)	指向缓冲区指针。
alenpp (IN/OUT)	用于读取绑定值后填充大小的指针。
piecep (IN/OUT)	仅做兼容。
indpp (OUT)	绑定列的类型指针。
rcoodepp (OUT)	返回内容的指针。

- 返回值

成功返回NCI\_SUCCESS，参数非法返回NCI\_INVALID\_HANDLE，失败返回NCI\_ERROR。

- 示例

```
NCIBindDynamic(bndhp, error, (dvoid *) &pos[0], cbf_no_data, (dvoid *) &pos[0], cbf_get_data);
```

### 3.5.13. NCIDefineDynamic

- 功能

如果在NCIDefineByPos中选择了NCI\_DYNAMIC\_FETCH模式，则调用此接口设置所需的附加属性。

- 函数

```
sword NCIDefineDynamic(NCIDefine *defnp,
NCIError *errhp,dvoid *octxp,
NCICallbackDefine(ocbfp)(/*_
dvoid *octxp,
NCIDefine *defnp,
ub4 iter,
dvoid **bufpp,
ub4 **alenpp,
ub1 *piecep,
dvoid **indpp,
ub2 **rcoodep_*/));
```

- 参数

表 3.36. NCIDefineDynamic 参数说明

名称	描述
defnp (IN/OUT)	定义信息句柄（通过调用NCIDefineByPos获得）。
errhp (IN/OUT)	错误句柄。
octxp (IN/OUT)	回调函数所需的上下文指针。
ocbfp (IN)	回调函数。
octxp (IN)	此回调函数的上下文指针。
defnp (IN)	定义的句柄。
iter (IN)	仅做兼容。
bufpp (OUT)	指向缓冲区的指针，用来存储列值。
alenpp (IN/OUT)	指向缓冲区指针。
piecep (IN/OUT)	仅做兼容。
indpp (OUT)	指针变量。
rccodep (IN)	仅做兼容。

- 返回值

成功返回NCI\_SUCCESS，参数非法返回NCI\_INVALID\_HANDLE，失败返回NCI\_ERROR。

- 示例

```
NCIDefineDynamic(defnp[1], error, (dvoid *) &res_buf, (NCICallbackDefine) cdf_fetch_buffer);
```

## 3.6. 事务

### 3.6.1. NCITransCommit

- 功能

提交与指定的服务上下文联系的事务。

- 函数

```
sword NCITransCommit(NCISvcCtx *svchp,
NCIError *errhp,
ub4 flags)
```

- 参数

表 3.37. NCITransCommit 参数说明

名称	描述
svchp（输入）	服务上下文句柄。
errhp（输入）	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。

名称	描述
flags（输入）	保留参数，用于全局事务中的单阶段提交优化的标志。

- 示例

```
NCITransCommit(svchpp, errhpp, 0);
```

### 3.6.2. NCITransRollback

- 功能

回滚当前事务。

- 函数

```
sword NCITransRollback(NCISvcCtx *svchp,
NCIError *errhp,
ub4 flags)
```

- 参数

表 3.38. NCITransRollback 参数说明

名称	描述
svchp（输入）	服务上下文句柄。
errhp（输入）	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
flags（输入）	保留参数。

- 示例

```
NCITransRollback(svchpp, errhpp, 0);
```

### 3.6.3. NCITransStart

- 功能

设置事务开始。

- 函数

```
sword NCITransStart(NCISvcCtx *svchp,
NCIError *errhp,
uword timeout,
ub4 flags)
```

- 参数

表 3.39. NCITransStart 参数说明

名称	描述
svchp（输入/输出）	服务上下文句柄。

名称	描述
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
timeout (输入)	保留参数。
falgs (输入)	<p>指定或者启动一个新的事务或者恢复一个现存的事务。并指定可串行的或只读的状态。可指定一个以上的值。默认情况是启动一个读/写事务。标记值如下所示。</p> <ul style="list-style-type: none"> <li>• NCI_TRANS_NEW: 启动一个新的事务分支。默认情况是启动一个紧耦合的并且可迁移的分支</li> <li>• NCI_TRANS_READONLY: 启动一个只读的事务。</li> <li>• NCI_TRANS_SERIALIZABLE: 启动一个可串行化的事务。</li> <li>• NCI_TRANS_RESUME: 唤醒一个事务。</li> </ul>

- 注意

此函数设置全局或可序列化事务的开始。如果flags参数指定应启动新事务，则当前与服务 上下文句柄关联的事务上下文在调用结束时初始化。

- 示例

```
NCITransStart(svchpp, errhpp, 60, NCI_TRANS_NEW);
```

### 3.6.4. NCITransDetach

- 功能

断开一个事务。

- 函数

```
sword NCITransDetach (NCISvcCtx *svchp,
NCIError *errhp,
ub4 flags)
```

- 参数

表 3.40. NCITransDetach 参数说明

名称	描述
svchp (输入/输出)	服务上下文句柄。
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
falgs (输入)	默认参数: NCI_DEFAULT。

- 示例



```
NCITransDetach(NULL, error, NCI_DEFAULT);
```

## 3.7. 大对象

### 3.7.1. NCILobRead

- 功能

读取LOB或者BFILE的一部分到缓冲区。

- 函数

```
sword NCILobRead(NCISvcCtx *svchp,  
NCIError *errhp,  
NCILobLocator *locp,  
ub4 *amtp,  
ub4 offset,  
dvoid *bufp,  
ub4 buf1,  
dvoid *ctxp,  
NCICallbackLobRead (cbfp)  
(dvoid *ctxp,  
CONST dvoid *bufp,  
ub4 len,  
ub1 piece),  
ub2 csid,  
ub1 csfrm)
```

- 参数

表 3.41. NCILobRead 参数说明

名称	描述
svchp (输入/输出)	服务上下文句柄。
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
locp (输入)	一个LOB句柄唯一引用LOB。此LOB句柄必须是从svchp指定的服务器获取的定位器。
amtp (输入/输出)	保留参数。
offset (输入)	在输入时，这是从LOB值开始的绝对偏移量。第一个位置是1。
buf1 (输入)	缓冲区的长度每次最多允许读1G（以八位字节为单位）。
ctxp (输入)	回调函数的上下文指针，可以为NULL。
cbfp (输入)	注册的回调函数，可为每个分片调用。若为空，则对每个分片将返回NCI_NEED_DATA。 当需要继续读取LOB数据时，回调函数返回值必须是NCI_CONTINUE。若返回错误，则 LOB的读将终止。回调函数具有固定的函数原型，以下为函数参数。

名称	描述
	<ul style="list-style-type: none"> <li>ctxp(输入): 回调函数的上下文, 可以为空。</li> <li>bufp(输入/输出): 指向LOB分片的数据缓冲区的指针。</li> <li>len(输入): bufp缓冲区中的当前分片的字节长度。</li> <li>piece(输入): 哪一个分片。</li> </ul>
csid (输入)	缓冲区数据的字符集ID。
csfrm (输入)	缓冲区数据的字符集表。这个参数必须要和LOB的类型一致。
bufp (输入/输出)	指向读取片段的缓冲区的指针。假定分配的内存长度为buf1。

表 3. 42. LOB数据字符数或者字节数

LOB	输入	输出定宽的客户端字符集	输出变宽的客户端字符集
BLOB	字节	字节	字节
CLOB	字符	字符	字节

输入数量指从服务器端CLOB读入的字符数。输出数量指读进缓冲区bufp的字节数。

- 示例

```
NCILobRead(svchp, errhp, lobp, &amtp, 1, buf+readsize, lenp+1-readsize, (dvoid *) 0, (sb4 *)
(dvoid *, const dvoid *, ub4, ub1))0, (ub2)0, (ub1)SQLCS_IMPLICIT);
```

### 3.7.2. NCILobWrite

- 功能

将缓冲区中的数据写入到LOB中。

- 函数

```
sword NCILobWrite(NCISvcCtx *svchp,
NCIError *errhp,
NCILobLocator *locp,
ub4 *amtp,
ub4 offset,
dvoid *bufp,
ub4 buflen,
ub1 piece,
dvoid *ctxp,
NCICallbackLobWrite (cbfp)
/*dvoid *ctxp,
dvoid *bufp,
```

```

ub4 *lenp,
ub1 *piecep */),
ub2 csid,
ub1 csfrm)

```

- 参数

表 3.43. NCILobWrite 参数说明

名称	描述
svchp (输入/输出)	服务上下文句柄。
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
locp (输入)	一个LOB句柄唯一引用LOB。此LOB句柄必须是从svchp指定的服务器获取的定位器。
amtp (输入/输出)	保留参数。
offset (输入)	在输入时，这是从LOB值开始的绝对偏移量。第一个位置是1。
bufp (输入)	指向读取片段的缓冲区的指针。假定分配的内存长度为buflen。
buflen (输入)	缓冲区的长度每次最多允许读1G（以八位字节为单位）。
piece (输入)	保留参数。
ctxp (输入)	回调函数的上下文。
cbfp (输入)	<p>在分片方式写入操作时对每个分片调用注册的回调函数。当cbfp为空时，将采用询问(polling)方式写入LOB数据。若需要写继续，则回调函数应该返回NCI_CONTINUE。否则将返回错误码，LOB写入操作将被终止。回调函数具有固定的函数原型，以下参数为该函数参数。</p> <ul style="list-style-type: none"> <li>• ctxp(输入)：回调函数上下文，可以为空。</li> <li>• bufp(输入/输出)：指向LOB分片数据缓冲区的指针。该参数与传递给 NCILobWrite() 函数的bufp相同。</li> <li>• lenp(输入/输出)：bufp缓冲区中LOB数据的字节长度(输入)，当前分片在bufp中的字节长度(输出)。</li> <li>• piecep(输出)：当前所写入哪一个数据分片。</li> </ul>
csid (输入)	缓冲区数据的字符集ID。
csfrm (输入)	缓冲区数据的字符集格式。csfrm参数必须要和LOB的类型一致。

表 3.44. LOB函数实际写入的LOB数据长度

LOB	输入定宽的客户端字符集	输入变宽的客户端字符集	输出
BLOB	字节	字节	字节
CLOB	字符	字节	字节

输入数量指用户要写进LOB的数据的字节数而不是在bufp中的字节数(由buflen)指定。

- 示例

```
NCILobWrite(svchp, errhp, lobj, &amtp, 0, (dvoid *)buf, (ub4)1024, 0, (dvoid *)0, 0, (ub2)0,
(ub1)SQLCS_IMPLICIT);
```

### 3.7.3. NCILobGetLength

- 功能

获取LOB的长度。

- 函数

```
sword NCILobGetLength(NCISvcCtx *svchp,
NCIError *errhp,
NCILobLocator *locp,
ub4 *lenp)
```

- 参数

表 3.45. NCILobGetLength 参数说明

名称	描述
svchp (输入)	服务上下文句柄。
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
locp (输入)	一个LOB句柄唯一引用LOB。此LOB句柄必须是从svchp指定的服务器获取的定位器。
lenp (输出)	在输出时, 如果LOB不为NULL, 则它是LOB的长度。

- 示例

```
NCILobGetLength(ph->phNCISvctx, ph->phNCIErr, lobl, &size);
```

### 3.7.4. NCILobCreateTemporary

- 功能

创建一个临时LOB对象。

- 函数

```

sword NCILobCreateTemporary(NCISvcCtx *svchp,
NCIError *errhp,
NCILobLocator *locp,
ub2 csid,
ub1 csfrm,
ub1 lobtype,
boolean cache,
NCIDuration duration)

```

- 参数

表 3. 46. NCILobCreateTemporary 参数说明

名称	描述
svchp (输入)	NCI服务上下文句柄。
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
locp (输入/输出)	指向临时大对象的定位器。
csid (输入)	保留参数。
csfrm (输入)	保留参数。
lobtype (输入)	创建的LOB类型，可选值如下所示。 <ul style="list-style-type: none"> <li>NCI_TEMP_BLOB：临时BLOB。</li> <li>NCI_TEMP_CLOB：临时CLOB。</li> </ul>
cache (输入)	保留参数。
duration (输入)	临时LOB对象的有效期限，可选值如下所示。 <ul style="list-style-type: none"> <li>NCI_DURATION_SESSION</li> <li>NCI_DURATION_CALL</li> </ul>

- 示例

```
NCILobCreateTemporary(svchp, errhp, &lob3, 0, 0, NCI_TEMP_BLOB, false, 0);
```

### 3. 7. 5. NCILobAssign

- 功能

赋值一个LOB对象到另一个。

- 函数

```

sword NCILobAssign(NCIEnv *envhp,
NCIError *errhp,
const NCILobLocator *src_locp,
NCILobLocator **dst_locpp)

```

- 参数

表 3.47. NCILobAssign 参数说明

名称	描述
envhp (输入/输出)	环境句柄。
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
src_locp (输入)	被赋值的LOB句柄。
dst_locpp (输入/输出)	需要设置的LOB句柄。

- 示例

```
NCILobAssign(envhp, errhp, NULL, NULL);
```

### 3.7.6. NCILobFreeTemporary

- 功能

释放一个临时LOB。

- 函数

```
sword NCILobFreeTemporary (NCISvcCtx *svchp,  
NCIError *errhp,  
NCILobLocator *locp)
```

- 参数

表 3.48. NCILobFreeTemporary 参数说明

名称	描述
svchp (输入/输出)	NCI服务上下文句柄。
errhp (输入/输出)	错误句柄，可以把它传给 NCIErrorGet() 以获得关于错误的详细信息。
locp(输入/输出)	LOB句柄，它唯一的指定了一个需要被释放的LOB。

- 示例

```
NCILobFreeTemporary (NULL, error, lob);
```

### 3.7.7. NCILobIsEqual

- 功能

判断给定的LOB句柄是否相等。

- 函数

```
sword NCILobIsEqual(NCIEnv *envhp,  
const NCILobLocator *x,  
const NCILobLocator *y,  
boolean *is_equal)
```

- 参数

表 3.49. NCILobIsEqual 参数说明

名称	描述
envhp（输入）	NCI环境句柄。
x（输入）	LOB句柄。
y（输入）	LOB句柄。
is_equal（输出）	TRUE表示相等，FALSE表示不等。

- 示例

```
NCILobIsEqual(envhp, &lob3, &lob4, &isEqual);
```

### 3.7.8. NCILobEnableBuffering

- 功能

开启Lob的缓存。

- 函数

```
sword NCILobEnableBuffering(  
    NCISvcCtx *svchp,  
    NCIError *err,  
    NCILobLocator *locp)
```

- 参数

表 3.50. NCILobEnableBuffering 参数说明

名称	描述
svchp（输入）	服务上下文句柄。
err（输入/输出）	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
locp（输入）	需要开启缓存的LOB句柄。

为LOB定位器启用LOB缓冲，之后通过该LOB读取或写入数据时，将使用LOB缓冲子系统。

- 注意

调用该接口后，该LOB调用这些接口时会返回错误，只有当调用NCILobDisableBuffering结束缓存功能后才能成功调用：  
NCILobAppend()、NCILobCopy()、NCILobErase()、NCILobGetLength()、NCILobTrim()。

- 示例

```
NCILobEnableBuffering(sc, error, clob);
```

### 3.7.9. NCILobDisableBuffering

- 功能

关闭Lob的缓存。

- 函数

```
sword NCILobDisableBuffering(
    NCISvcCtx *svchp,
    NCIError *err,
    NCILobLocator *locp)
```

- 参数

表 3.51. NCILobDisableBuffering 参数说明

名称	描述
svchp (输入)	服务上下文句柄。
err (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
locp (输入)	需要关闭缓存的LOB句柄。

- 示例

```
NCILobDisableBuffering(svcctx, error, clob);
```

### 3.7.10. NCILobFlushBuffer

- 功能

将此 LOB 的所有缓冲区刷新或写入服务器。

- 函数

```
sword NCILobFlushBuffer(
    NCISvcCtx *svchp,
    NCIError *err,
    NCILobLocator *locp,
    ub4 flag)
```

- 参数

表 3.52. NCILobFlushBuffer 参数说明

名称	描述
svchp (输入)	服务上下文句柄。
err (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
locp (输入)	需要写入的LOB句柄。
flag (输入)	缓冲区释放标识，如下所示。 <ul style="list-style-type: none"> <li>• NCI_LOB_BUFFER_FREE: 释放缓冲区。</li> <li>• NCI_LOB_BUFFER_NOFREE: 不释放缓冲区，仅重置缓冲区内容。</li> </ul>



- 示例

```
NCILobFlushBuffer(svcctx, error, clob, NCI_LOB_BUFFER_NOFREE);
```

### 3.7.11. NCILobAppend

- 功能

在另一个 LOB 的末尾附加一个 LOB 值。

- 函数

```
sword NCILobAppend (
    NCISvcCtx      *svchp,
    NCIError       *errhp,
    NCILobLocator  *dst_locp,
    NCILobLocator  *src_locp );
```

- 参数

表 3.53. NCILobAppend 参数说明

名称	描述
svchp (输入)	服务上下文句柄。
errhp (输入/输出)	错误句柄。
dst_locp (输入/输出)	被附加的LOB句柄。
src_locp (输入)	附加的LOB句柄。

- 示例

```
NCILobAppend(svcctx, error, lob_dest, lob_src);
```

### 3.7.12. NCILobTrim

- 功能

将 LOB 值截断。

- 函数

```
sword NCILobTrim (
    NCISvcCtx      *svchp,
    NCIError       *errhp,
    NCILobLocator  *locp,
    ub4            newlen );
```

- 参数

表 3.54. NCILobTrim 参数说明

名称	描述
svchp (输入)	服务上下文句柄。
errhp (输入/输出)	错误句柄。

名称	描述
locp（输入/输出）	被截断的LOB句柄。
newlen（输入）	新LOB的长度。

- 示例

```
NCILobTrim(svcctx, error, lob, newlen);
```

### 3.7.13. NCILobErase

- 功能

从指定偏移量开始删除内部 LOB 数据的指定部分。

- 函数

```
sword NCILobErase (
    NCISvcCtx    *svchp,
    NCIError     *errhp,
    NCILobLocator *locp,
    ub4          *amount,
    ub4          offset );
```

- 参数

表 3.55. NCILobErase 参数说明

名称	描述
svchp（输入）	服务上下文句柄。
errhp（输入/输出）	错误句柄。
locp（输入/输出）	指定的LOB句柄。
amount（输入/输出）	需要删除的字节数。IN的时候为需要删除的字节数，OUT的时候为实际删除的字节数。
offset（输入）	绝对偏移量，从1开始。

- 注意

被删除的字节会被填充，并不会真正删除，CLOB类型会被填充为空格，BLOB类型会被填充为\0。

- 示例

```
NCILobErase(svcctx, error, lob, &amount, offset);
```

### 3.7.14. NCILobCopy

- 功能

将一个 LOB 值的全部或部分复制到另一个 LOB 值中。

- 函数

```
sword NCILobCopy (
```

```

NCISvcCtx    *svchp,
NCIError     *errhp,
NCILobLocator *dst_locp,
NCILobLocator *src_locp,
ub4          amount,
ub4          dst_offset,
ub4          src_offset );

```

- 参数

表 3.56. NCILobCopy 参数说明

名称	描述
svchp (输入)	服务上下文句柄。
errhp (输入/输出)	错误句柄。
dst_locp (输入/输出)	目的LOB句柄。
src_locp (输入)	源LOB句柄。
amount (输入)	源LOB句柄需要复制的字节数。
dst_offset (输入)	目的LOB句柄复制开始的偏移量。
src_offset (输入)	源LOB句柄复制的偏移量。

- 注意

如果拷贝的长度大于大对象的长度，最多就拷贝大对象的长度。

- 示例

```
NCILobCopy(svcctx, error, lob_dest, lob_src, amount, dst_offset, src_offset);
```

### 3.7.15. NCILobOpen

- 功能

以指定模式打开一个大对象。

- 函数

```

sword NCILobOpen (
NCISvcCtx    *svchp,
NCIError     *errhp,
NCILobLocator *locp,
ub1          mode );

```

- 参数

表 3.57. NCILobOpen 参数说明

名称	描述
svchp (输入)	服务上下文句柄。
errhp (输入/输出)	错误句柄。
locp (输入/输出)	要打开的LOB句柄。

名称	描述
mode（输入）	打开模式。 <ul style="list-style-type: none"><li>• NCI_LOB_READONLY：仅读。</li><li>• NCI_LOB_WRITEONLY：仅写。</li><li>• NCI_LOB_APPENDONLY：写且将lob指针移到最后。</li><li>• NCI_LOB_READWRITE：读写。</li></ul>

- 示例

```
NCILobOpen(svcctx, error, lob, NCI_LOB_READONLY);
```

### 3.7.16. NCILobClose

- 功能

关闭一个大对象。

- 函数

```
sword NCILobClose (  
    NCISvcCtx    *svchp,  
    NCIError     *errhp,  
    NCILobLocator *locp);
```

- 参数

表 3.58. NCILobClose 参数说明

名称	描述
svchp（输入）	服务上下文句柄。
errhp（输入/输出）	错误句柄。
locp（输入/输出）	需要关闭的LOB句柄。

- 示例

```
NCILobClose(svcctx, error, lob);
```

### 3.7.17. NCILobLocatorIsInit

- 功能

LOB是否被初始化。

- 函数

```
sword NCILobLocatorIsInit(  
    NCIEnv *envhp,  
    NCIError *errhp,  
    CONST NCILobLocator *locp,
```

```
Boolean *is_initialized );
```

- 参数

表 3.59. NCILobLocatorIsInit 参数说明

名称	描述
envhp (输入)	环境句柄。
errhp (输入/输出)	错误句柄。
locp (输入)	判断的Lob句柄。
is_initialized (输出)	是否被初始化。

- 示例

```
NCILobLocatorIsInit(envhp, errhp, locp, &is_init);
```

### 3.7.18. NCILobLocatorAssign

- 功能

赋值一个LOB至另一个。

- 函数

```
sword NCILobLocatorAssign (
    NCISvcCtx *svchp,
    NCIError *errhp,
    const NCILobLocator *src_locp,
    NCILobLocator **dst_locpp);
```

- 参数

表 3.60. NCILobLocatorAssign 参数说明

名称	描述
svchp (输入/输出)	语句句柄。
errhp (输入/输出)	错误句柄。
src_locp (输入)	被赋值的LOB句柄。
dst_locp (输入/输出)	需要设置的LOB句柄。必须通过调用 NCIDescriptorAlloc() 分配空间。

- 示例

```
NCILobLocatorAssign(svchp, errhp, src_locp, &dst_locp);
```

## 3.8. RAW对象

### 3.8.1. NCIRawPtr

- 功能

获取指向原始数据的指针。

- 函数

```
sword NCIRawPtr (NCIEnv *envhp,  
const NCIRaw* raw)
```

- 参数

表 3.61. NCIRawPtr 参数说明

名称	描述
envhp（输入）	NCI环境句柄。
raw（输入）	Raw对象指针。

- 示例

```
NCIRawPtr(env, NULL);
```

### 3.8.2. NCIRawSize

- 功能

返回raw对象原始数据的大小。

- 函数

```
sword NCIRawPtr (NCIEnv *envhp,  
const NCIRaw* raw)
```

- 参数

表 3.62. NCIRawSize 参数说明

名称	描述
envhp（输入）	NCI环境句柄。
raw（输入）	Raw对象指针。

- 示例

```
NCIRawSize(NULL, raw);
```

### 3.8.3. NCIRawAllocSize

- 功能

获取Raw对象分配的大小。

- 函数

```
sword NCIRawAllocSize (NCIEnv* envhp,  
NCIError *errhp,
```

```
const NCIRaw* raw)
ub4* allocsize);
```

- 参数

表 3.63. NCIRawAllocSize 参数说明

名称	描述
envhp (输入)	NCI环境句柄。
errhp (输入/输出)	错误句柄，可以把它传给 NCIErrorGet() 以获得关于错误的详细信息。
raw(输入)	Raw对象指针。
allocsize(输出)	返回raw对象分配的大小。

- 示例

```
NCIRawAllocSize(env, error, raw, &allocsize);
```

### 3.8.4. NCIRawAssignBytes

- 功能

将一段内存复制给一个raw对象。

- 函数

```
sword NCIRawAssignBytes (NCIEnv* envhp,
NCIError *errhp,
const ub1* source,
ub4 source_len,
NCIRaw** target)
```

- 参数

表 3.64. NCIRawAssignBytes 参数说明

名称	描述
envhp (输入)	NCI环境句柄。
errhp (输入/输出)	错误句柄，可以把它传给NCIErrorGet() 以获得关于错误的详细信息。
source (输入)	需要复制的内存地址。
source_len(输入)	需要复制的内存的长度。
target (输出)	返回的raw对象。

- 示例

```
NCIRawAssignBytes(env, NULL, data, len, &raw);
```

### 3.8.5. NCIRawResize

- 功能

重新分配raw对象的大小。

- 函数

```
sword NCIRawResize (NCIEnv* envhp,  
    NCIError *errhp,  
    ub2 new_size,  
    NCIRaw** rawp)
```

- 参数

表 3.65. NCIRawResize 参数说明

名称	描述
envhp (输入)	NCI环境句柄。
errhp (输入/输出)	错误句柄，可以把它传给NCIErrorGet()以获得关于错误的详细信息。
new_size (输入)	重新分配的大小。
Rawp (输出)	重新分配过大小的Raw对象。

- 示例

```
NCIRawResize(NULL, error, new_size, &raw);
```

### 3.8.6. NCIRawAssignRaw

- 功能

通过一个raw对象复制出另一个raw对象。

- 函数

```
sword NCIRawAssignRaw (NCIEnv* envhp,  
    NCIError *errhp,  
    const NCIRaw* rhs,  
    NCIRaw** lhs)
```

- 参数

表 3.66. NCIRawAssignRaw 参数说明

名称	描述
svchp (输入)	NCI环境句柄。
errhp (输入/输出)	错误句柄，可以把它传给 NCIErrorGet() 以获得关于错误的详细信息。
rhs (输入)	需要复制的Raw对象。
lhs (输出)	复制过后生成的Raw对象。

- 示例

```
NCIRawAssignRaw(env, NULL, raw_rhs, &raw_lhs);
```



## 3.9. 时间

### 3.9.1. NCIDateTimeConstruct

- 功能

初始化datetime描述符的信息。

- 函数

```
sword NCIDateTimeConstruct(  
    void *hdl,  
    NCIError *err,  
    NCIDateTime *datetime,  
    sb2 yr,  
    ub1 mnth,  
    ub1 dy,  
    ub1 hr,  
    ub1 mm,  
    ub1 ss,  
    ub4 fsec,  
    OraText *timezone,  
    size_t timezone_length)
```

- 参数

表 3.67. NCIDateTimeConstruct 参数说明

名称	描述
hdl（输入）	NCI环境句柄。
err（输入/输出）	错误句柄。
datetime（输入）	NCIDateTime的描述符指针。
yr（输入）	年。
mnth（输入）	月。
dy（输入）	日。
hr（输入）	时。
mm（输入）	分。
ss（输入）	秒。
fsec（输入）	小数秒。
timezone（输入）	时区。
timezone_length（输入）	时区长度。

日期时间参数范围请参见《优炫数据库 2.1 参考手册》“数据类型”章节中的“日期/时间类型”小节中的“日期/时间类型”表。

- 注意

datetime参数需要先通过NCIDescriptorAlloc函数分配空间。

- 示例

```
NCIDDescriptorAlloc((dvoid *)env,(void **)&constructdatetime, NCI_DTYPE_DATE, 0, (void
***)0);
```

### 3.9.2. NCIDateTimeFromText

- 功能

把字符串按照一定格式转换成DateTime类型。

- 函数

```
sword NCIDateTimeFromText(
dvoid *hdl,
NCIError *err,
const OraText *date_str,
size_t dstr_length,
const OraText *fmt,
ub1 fmt_length,
const OraText *lang_name,
size_t lang_length,
NCIDateTime *datetime)
```

- 参数

表 3.68. NCIDateTimeFromText 参数说明

名称	描述
hdl（输入）	NCI环境句柄。
err（输入/输出）	错误句柄。
date_str（输入）	用来转换的日期字符串。
dstr_length（输入）	日期字符串的长度。
fmt（输入）	日期格式串。
fmt_length（输入）	日期格式串长度。
lang_name（输入）	日期中语言的名称。
lang_length（输入）	语言字符串的长度。
datetime（输出）	NCIDateTime类型指针。

日期时间参数范围请参见《优炫数据库 2.1 参考手册》“函数和操作符”章节中的“数据类型格式化函数”小节中的“用于日期/时间格式化的模板模式”表。

- 注意

datetime参数需要先通过NCIDDescriptorAlloc函数分配空间。日期中的语言使用的是默认数据库中的语言，暂时不支持按照输入的语言转换。

- 示例

```
NCIDateTimeFromText((dvoid *)env, error, column_time, sizeof(column_time), fmt, sizeof(fmt),
(OraText *)LANG, sizeof(LANG), datetime);
```

### 3.9.3. NCIDateTimeToText

- 功能

根据指定格式把一个日期类型数据转换成字符串。

- 函数

```
sword NCIDateTimeToText(
    dvoid *hdl,
    NCLError *err,
    NCIDateTime *datetime,
    const OraText *fmt,
    ubl fmt_length,
    ubl fsprec,
    const OraText *lang_name,
    size_t lang_length,
    size_t *buf_size,
    const OraText *buf)
```

- 参数

表 3.69. NCIDateTimeToText 参数说明

名称	描述
hdl (输入)	NCI环境句柄。
err (输入/输出)	错误句柄。
datetime (输入)	用来转换的NCIDateTime类型指针。
fmt (输入)	日期格式串。
fmt_length (输入)	日期格式串长度。
fsprec (输入)	指定返回结果字符串中秒的精度。
lang_name (输入)	日期中语言的名称。
lang_length (输入)	语言字符串的长度。
buf_size (输入\输出)	结果字符串的长度。
buf (输出)	缓冲区指针，转换后字符串存入该区域。

日期时间参数范围请参见《优炫数据库 2.1 参考手册》“函数和操作符”章节中的“数据类型格式化函数”小节中的“用于日期/时间格式化的模板模式”表。

- 注意

日期中的语言使用的是默认数据库中的语言，暂时不支持按照输入的语言转换。秒的精度暂时不支持按照输入的计算，默认是6位小数。

- 示例

```
NCIDateTimeToText((dvoid *)env, error, datetime, fmt, strlen((char*)fmt), 2, NULL, 0, &buf_size,
    timeout);
```

### 3.9.4. NCIDateTimeGetDate

- 功能

获取NCIDateTime中的日期信息。

- 函数

```
sword NCIDateTimeGetDate(  
    dvoid *hdl,  
    NCIError *err,  
    CONST NCIDateTime *datetime,  
    sb2 *yr,  
    ub1 *mnth,  
    ub1 *dy)
```

- 参数

表 3.70. NCIDateTimeGetDate 参数说明

名称	描述
hdl (输入)	NCI环境句柄。
err (输入/输出)	错误句柄。
datetime (输入)	要获取的NCIDateTime描述符指针。
yr (输出)	年。
mnth (输出)	月。
dy (输出)	日。

- 注意

如果datetime是NCI\_DTYPE\_TIME或NCI\_DTYPE\_TIME\_TZ类型，获取日期会返回错误NCI\_ERROR。

- 示例

```
NCIDateTimeGetDate((dvoid *)env, error, datetime, &year, &month, &day);
```

### 3.9.5. NCIDateTimeGetTime

- 功能

初始化datetime描述符的信息。

- 函数

```
sword NCIDateTimeGetTime(  
    void *hdl,  
    NCIError *err,  
    NCIDateTime *datetime,  
    ub1 *hr,  
    ub1 *mm,  
    ub1 *ss,  
    ub4 *fsec)
```

- 参数

表 3.71. NCIDateTimeGetTime 参数说明

名称	描述
hdl (输入)	NCI环境句柄。
err (输入/输出)	错误句柄。
datetime (输入)	要获取的NCIDateTime描述符指针。
hr (输出)	时。
mm (输出)	分。
ss (输出)	秒。
fsec (输出)	小数秒。

- 注意

如果datetime是NCI\_DTYPE\_DATE类型，获取时间会返回错误NCI\_ERROR。

- 示例

```
NCIDateTimeGetTime((dvoid *)env, error, datetime, &hr, &mm, &ss, &fsec);
```

## 3.10. 时间间隔

时间间隔句柄需要事先调用NCIDescrptorAlloc申请空间，事后调用NCIDescrptorFree释放空间。

### 3.10.1. NCIIIntervalSetYearMonth

- 功能

设置时间间隔的年月。

- 函数

```
sword NCIIIntervalSetYearMonth(
    void *hdl,
    NCIError *err,
    sb4 yr,
    sb4 mnth,
    NCIInterval *result)
```

- 参数

表 3.72. NCIIIntervalSetYearMonth 参数说明

名称	描述
hdl (输入)	NCI环境句柄。
err (输入/输出)	错误句柄。
yr (输入)	年。
mnth (输入)	月。
result (输出)	输出的时间间隔。

- 注意

设置成功后天秒部分会被清空。

- 示例

```
NCIIntervalSetYearMonth(env, error, 1, 2, inter);
```

### 3.10.2. NCIIntervalGetYearMonth

- 功能

获取时间间隔的年月。

- 函数

```
sword NCIIntervalGetYearMonth(
    void *hdl,
    NCIError *err,
    sb4 *yr,
    sb4 *mnth,
    const NCIInterval *interval)
```

- 参数

表 3.73. NCIIntervalGetYearMonth 参数说明

名称	描述
hdl（输入）	NCI环境句柄。
err（输入/输出）	错误句柄。
yr（输入）	年，范围为-178000000到178000000。
mnth（输入）	月。
interval（输出）	需要获取年月的时间间隔。

- 示例

```
NCIIntervalGetYearMonth(env, error, &year, &month, inter);
```

### 3.10.3. NCIIntervalSetDaySecond

- 功能

设置时间间隔的天秒。

- 函数

```
sword NCIIntervalSetDaySecond(
    void *hdl,
    NCIError *err,
    sb4 dy,
    sb4 hr,
    sb4 mm,
    sb4 ss,
```

```
sb4 fsec,  
NCIInterval *result)
```

- 参数

表 3.74. NCIIntervalSetDaySecond 参数说明

名称	描述
hdl (输入)	NCI环境句柄。
err (输入/输出)	错误句柄。
dy (输入)	天。
hr (输入)	小时。
mm (输入)	分钟。
ss (输入)	秒。
fsec (输入)	微秒。
result (输出)	输出的时间间隔。

- 注意

设置成功后年月部分会被清空。

- 示例

```
NCIIntervalSetDaySecond(NULL, error, 1, 2, 3, 4, 5, inter);
```

### 3.10.4. NCIIntervalGetDaySecond

- 功能

获取时间间隔的天秒。

- 函数

```
sword NCIIntervalGetDaySecond(  
    void *hdl,  
    NCIError *err,  
    sb4 *dy,  
    sb4 *hr,  
    sb4 *mm,  
    sb4 *ss,  
    sb4 *fsec,  
    NCIInterval *interval)
```

- 参数

表 3.75. NCIIntervalGetDaySecond 参数说明

名称	描述
hdl (输入)	NCI环境句柄。
err (输入/输出)	错误句柄。
dy (输入)	天。

名称	描述
hr（输入）	小时。
mm（输出）	分钟。
ss（输出）	秒。
fsec（输出）	微秒。
interval（输入）	需要获取时间的时间间隔。

- 示例

```
NCIIntervalGetDaySecond(env, error, &dd, &hh, &mm, &ss, &us, inter);
```

### 3.10.5. NCIIntervalCompare

- 功能

比较两个时间间隔。

- 函数

```
sword NCIIntervalCompare(  
    void *hdl,  
    NCIError *err,  
    NCIInterval *inter1,  
    NCIInterval *inter2,  
    sword *result)
```

- 参数

表 3.76. NCIIntervalCompare 参数说明

名称	描述
hdl（输入）	NCI环境句柄。
err（输入/输出）	错误句柄。
inter1（输入）	时间间隔1。
inter2（输入）	时间间隔2。
result（输出）	比较结果。 <ul style="list-style-type: none"><li>• 如果inter1 &gt; inter2, 则result = 1。</li><li>• 如果inter1 = inter2, 则result = 0。</li><li>• 如果inter1 &lt; inter2, 则result = -1。</li></ul>

- 示例

```
NCIIntervalCompare(NULL, error, inter1, inter2, &result);
```

### 3.10.6. NCIIntervalAdd

- 功能



两个时间间隔相加。

- 函数

```
sword NCIntervalAdd (
    void *hdl,
    NCLError *err,
    NCInterval *addend1,
    NCInterval *addend2,
    NCInterval *result)
```

- 参数

表 3.77. NCIntervalAdd 参数说明

名称	描述
hdl (输入)	NCI环境句柄。
err (输入/输出)	错误句柄。
addend1 (输入)	时间间隔1。
addend2 (输入)	时间间隔2。
result (输出)	相加的结果。

- 示例

```
NCIntervalAdd(env, error, inter1, inter2, inter3);
```

### 3.10.7. NCIntervalAssign

- 功能

将一个时间间隔赋值给另一个时间间隔。

- 函数

```
sword NCIntervalAssign(
    void *hdl,
    NCLError *err,
    NCInterval *ininter,
    NCInterval *outinter)
```

- 参数

表 3.78. NCIntervalAssign 参数说明

名称	描述
hdl (输入)	NCI环境句柄。
err (输入/输出)	错误句柄。
ininter (输入)	输入的时间间隔。
outinter (输出)	输出的时间间隔。

- 示例

```
NCIIntervalAssign(env, error, inter1, inter2);
```

### 3.10.8. NCIIntervalToText

- 功能

将时间间隔转成字符串。

- 函数

```
sword NCIIntervalToText(
    dvoid *hdl,
    NCIError *err,
    CONST NCIInterval *interval,
    ub1 lfprec,
    ub1 fsprec,
    OraText *buffer,
    size_t buflen,
    size_t *resultlen)
```

- 参数

表 3.79. NCIIntervalToText 参数说明

名称	描述
hdl (输入)	NCI环境句柄。
err (输入/输出)	错误句柄。
interval (输入)	时间间隔。
lfprec (输入)	阈值达到的精度(仅做兼容)。
fsprec (输入)	间隔中分秒的精度(仅做兼容)。
buffer (输出)	用于保存结果的缓冲区。
buflen (输入)	缓冲区长度。
resultlen (输出)	存放缓冲区buffer中结果的长度。

- 示例

```
NCIIntervalToText(env, error, inter, 0, 0, (OraText*)buf, sizeof(buf), &reslen);
```

### 3.10.9. NCIIntervalFromText

- 功能

将字符串转成时间间隔。

- 函数

```
sword NCIIntervalFromText(
    void *hdl,
    NCIError *err,
    const OraText *inpstr,
```

```
size_t str_len,
NCIInterval *result)
```

- 参数

表 3.80. NCIIntervalFromText 参数说明

名称	描述
hndl（输入）	NCI环境句柄。
err（输入/输出）	错误句柄。
inpstr（输入）	时间间隔字符串。允许的输入字符串参考请参考《优炫数据库 2.1 参考手册》“数据类型”章节中的“日期/时间类型”小节中的“间隔输入”。
str_len（输入）	时间间隔字符串长度。
result（输出）	转换后的时间间隔。

- 示例

```
NCIIntervalFromText(env, error, (OraText*)"10year 1day 2hour", 17, inter);
```

### 3.10.10. NCIIntervalCheck

- 功能

设置时间间隔的年月。

- 函数

```
sword NCIIntervalCheck(
    void *hndl,
    NCIError *err,
    NCIInterval *interval,
    ub4 *valid);
```

- 参数

表 3.81. NCIIntervalCheck 参数说明

名称	描述
hndl（输入）	NCI环境句柄。
err（输入/输出）	错误句柄。
interval（输入）	要检查的时间间隔。
valid（输入）	检查结果。

- 注意

这个函数仅检查输入的时间间隔句柄是否合法。

- 示例

```
NCIIntervalCheck(env, error, inter, &valid);
```

### 3.10.11. NCIntervalDivide

- 功能

利用数字划分一个时间间隔来产生一个时间间隔。

- 函数

```
sword NCIntervalDivide (  
    dvoid *hdl,  
    NCError *err,  
    NCInterval *dividend,  
    NCINumber *divisor,  
    NCInterval *result );
```

- 参数

表 3.82. NCIntervalDivide 参数说明

名称	描述
hdl (输入)	NCI环境句柄。
err (输入/输出)	错误句柄。
dividend (输入)	被划分的间隔。
divisor (输入)	用于划分的数字。
result (输出)	结果间隔(dividend/divisor)。

- 示例

```
NCIntervalDivide(env, error, inter1, num, result);
```

### 3.10.12. NCIntervalMultiply

- 功能

利用数字增长间隔以得到一个间隔。

- 函数

```
sword NCIntervalMultiply (  
    dvoid *hdl,  
    NCError *err,  
    CONST NCInterval *inter,  
    NCINumber *nfactor,  
    NCInterval *result);
```

- 参数

表 3.83. NCIntervalMultiply 参数说明

名称	描述
hdl (输入)	NCI环境句柄。
err (输入/输出)	错误句柄。

名称	描述
inter (输入)	要被增长的间隔。
nfactor	用于增长的数字，即增长量倍数。
result	结果间隔(inter * nfactor)。

- 示例

```
NCIIntervalMultiply(env, error, inter1, num, result);
```

### 3.10.13. NCIIntervalSubtract

- 功能

两个间隔相减。

- 函数

```
sword NCIIntervalSubtract (
    dvoid *hdl,
    NCIError *err,
    NCIInterval *minuend,
    NCIInterval *subtrahend,
    NCIInterval *result );
```

- 参数

表 3.84. NCIIntervalSubtract 参数说明

名称	描述
hdl (输入)	NCI环境句柄。
err (输入/输出)	错误句柄。
minuend (输入)	被减数间隔。
subtrahend (输入)	减数间隔。
result (输出)	结果间隔(minuend - subtrahend)。

- 示例

```
NCIIntervalSubtract(env, error, inter1, inter2, result);
```

## 3.11. 数字

### 3.11.1. NCINumberFromInt

- 功能

从int类型转成NCINumber类型。

- 函数

```
sword NCINumberFromInt (
    NCIError *err,
    const void *inum,
```

```
uword inum_length,
uword inum_s_flag,
NCINumber *number)
```

- 参数

表 3.85. NCINumberFromInt 参数说明

名称	描述
err (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
inum (输入)	需要转换的整形指针。
inum_length (输入)	整型长度，如下所示。 <ul style="list-style-type: none"> <li>• 1: inum类型为sb1或ub1。</li> <li>• 2: inum类型为sb2或ub2。</li> <li>• 4: inum类型为sb4或ub4。</li> </ul>
inum_s_flag (输入)	转换类型，如下所示。 <ul style="list-style-type: none"> <li>• NCI_NUMBER_SIGNED: 无符号类型。</li> <li>• NCI_NUMBER_UNSIGNED: 有符号类型。</li> </ul>
number (输出)	转换后的NCINumber类型，需要提前使用NCIDestructorAlloc分配空间。

- 示例

```
NCINumberFromInt(error, inum, sizeof(inum), NCI_NUMBER_UNSIGNED, number_int);
```

### 3.11.2. NCINumberToInt

- 功能

从NCINumber类型转化成整型。

- 函数

```
sword NCINumberToInt (
NCIError *err,
const NCINumber *number,
uword rsl_length,
uword rsl_s_flag,
void *rsl)
```

- 参数

表 3.86. NCINumberToInt 参数说明

名称	描述
err (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。

名称	描述
number (输入)	需要转换的NCINumber类型。
rsl_length (输入)	转换结果的长度，如下所示。 <ul style="list-style-type: none"> <li>1: rsl类型为sb1或ub1。</li> <li>2: rsl类型为sb2或ub2。</li> <li>4: rsl类型为sb4或ub4。</li> </ul>
rsl_flag (输入)	转换类型，如下所示。 <ul style="list-style-type: none"> <li>NCI_NUMBER_SIGNED: 无符号类型。</li> <li>NCI_NUMBER_UNSIGNED: 有符号类型。</li> </ul>
rsl (输出)	转换结果。

- 示例

```
NCINumberToInt(error, number_int, sizeof(inum), NCI_NUMBER_UNSIGNED, &inum);
```

### 3.11.3. NCINumberFromText

- 功能

从字符串转成NCINumber类型。

- 函数

```
sword NCINumberFromText(
    NCLError *err,
    CONST OraText *str,
    ub4 str_length,
    CONST OraText *fmt,
    ub4 fmt_length,
    CONST OraText *nls_params,
    ub4 nls_p_length,
    NCINumber *number )
```

- 参数

表 3.87. NCINumberFromText 参数说明

名称	描述
err (输入/输出)	出现错误时可以传递给NCLErrorGet() 诊断信息的错误句柄。
str (输入)	需要转换的字符串。
str_length (输入)	需要转换的字符串的长度。
fmt (输入)	转换格式，保留参数。
fmt_length (输入)	转换格式的长度，保留参数。
nls_params (输入)	间隔格式，保留参数。
nls_p_length (输入)	间隔格式的长度，保留参数。

名称	描述
number (输出)	转换后的NCINumber类型，需要提前使用NCIDescriptorAlloc分配空间。

- 示例

```
NCINumberFromText(error, inum, strlen((char*)inum), 0, 0, 0, number_int);
```

### 3.11.4. NCINumberToText

- 功能

NCINumber类型转成字符串类型。

- 函数

```
sword NCINumberToText(
    NCLError *err,
    const NCINumber *number,
    const oratext *fmt,
    ub4 fmt_length,
    const oratext *nls_params,
    ub4 nls_p_length,
    ub4 *buf_size,
    oratext *buf)
```

- 参数

表 3.88. NCINumberToText 参数说明

名称	描述
err (输入/输出)	出现错误时可以传递给NCLErrorGet() 诊断信息的错误句柄。
number (输入)	需要转换的NCINumber类型。
fmt (输入)	转换格式，保留参数。
fmt_length (输入)	转换格式长度，保留参数。
nls_params (输入)	间隔格式，保留参数。
nls_p_length (输入)	间隔格式的长度，保留参数。
buf_size (输入/输出)	字符串长度。
buf (输出)	字符串。

- 示例

```
NCINumberToText(error, number_int, NULL, 0, NULL, 0, &length, OraTextout);
```

### 3.11.5. NCINumberFromReal

- 功能

从浮点数转换成NCINumber类型。



- 函数

```
sword NCINumberFromReal(
    NCLError *err,
    const void *rnum,
    uword rnum_length,
    NCINumber *number)
```

- 参数

表 3.89. NCINumberFromReal 参数说明

名称	描述
err (输入/输出)	出现错误时可以传递给NCLErrorGet() 诊断信息的错误句柄。
rnum (输入)	浮点数指针。
rnum_length (输入)	浮点数指针长度，如下所示。 <ul style="list-style-type: none"> <li>• 4: rnum类型为float。</li> <li>• 8: rnum类型为double。</li> </ul>
number (输出)	转换后的NCINumber类型，需要提前使用NCIDescriptorAlloc分配空间。

- 示例

```
NCINumberFromReal(error, (void *)&dbl, sizeof(dbl), nres);
```

### 3.11.6. NCINumberToReal

- 功能

将NCINumber的值转换成浮点型的数值。

- 函数

```
sword NCINumberToReal(
    NCLError *err,
    const NCINumber *number,
    uword rsl_length,
    void *rsl)
```

- 参数

表 3.90. NCINumberToReal 参数说明

名称	描述
err (输入/输出)	出现错误时可以传递给NCLErrorGet() 诊断信息的错误句柄。
number (输入)	NCINumber类型指针。
rsl_length (输入)	rsl的长度，如下所示。 <ul style="list-style-type: none"> <li>• 4: rnum类型为float。</li> </ul>

名称	描述
	• 8: rnum类型为double。
rsl (输出)	浮点型指针。

- 示例

```
NCINumberToReal(error, nres, sizeof(dbl), (void*)&dbl);
```

### 3.11.7. NCINumberAdd

- 功能

两个NCINumber类型相加。

- 函数

```
sword NCINumberAdd(
    NCLError *err,
    const NCINumber *number1,
    NCINumber *result)
```

- 参数

表 3.91. NCINumberAdd 参数说明

名称	描述
err (输入/输出)	出现错误时可以传递给NCLErrorGet() 诊断信息的错误句柄。
number1 (输入)	被加数。
number2 (输入)	加数。
result (输出)	和，需要提前使用NCIDescriptorAlloc分配空间。

- 示例

```
NCINumberAdd(error, number_int, number_int2, result);
```

### 3.11.8. NCINumberAssign

- 功能

对Number进行赋值。

- 函数

```
sword NCINumberAssign(
    NCLError *err,
    const NCINumber *from,
    const NCINumber *to)
```

- 参数

表 3.92. NCINumberAssign 参数说明

名称	描述
err (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
from (输入)	源Number。
to (输出)	目标Number, 需要提前使用NCIDescriptorAlloc分配空间。

- 示例

```
NCINumberAssign(error, number_int, result);
```

### 3.11.9. NCINumberDiv

- 功能

计算两个Number的商。

- 函数

```
sword NCINumberDiv(NCError *err,  
const NCINumber *number1,  
const NCINumber *number2,  
NCINumber *result)
```

- 参数

表 3.93. NCINumberDiv 参数说明

名称	描述
err (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
number1 (输入)	分子。
number2 (输入)	分母。
result (输出)	商, 需要提前使用NCIDescriptorAlloc分配空间。

- 示例

```
NCINumberDiv(error, number_input, number_input2, result);
```

### 3.11.10. NCINumberIsInt

- 功能

判断Number类型是不是一个整数。

- 函数

```
sword NCINumberIsInt(
```

```
NCIError *err,  
const NCINumber *number,  
boolean *result)
```

- 参数

表 3.94. NCINumberIsInt 参数说明

名称	描述
err（输入/输出）	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
number（输入）	待判断数。
result（输出）	判断结果，TRUE是整数，FALSE不是整数。

- 示例

```
NCINumberIsInt(error, nres, &result);
```

### 3.11.11. NCINumberMod

- 功能

计算两个Number类型的余数。

- 函数

```
sword NCINumberMod(  
NCIError *err,  
const NCINumber *number1,  
const NCINumber *number2,  
NCINumber *result)
```

- 参数

表 3.95. NCINumberMod 参数说明

名称	描述
err（输入/输出）	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
number1（输入）	分子。
number2（输入）	分母。
result（输出）	结果，需要提前使用NCIDescriptorAlloc分配空间。

- 示例

```
NCINumberMod(error, number_int, number_int2, result);
```

### 3.11.12. NCINumberMul

- 功能

计算两个Number的积。

- 函数

```
sword NCINumberMul(  
    NCLError *err,  
    const NCINumber *number1,  
    NCINumber *result)
```

- 参数

表 3.96. NCINumberMul 参数说明

名称	描述
err（输入/输出）	出现错误时可以传递给NCLErrorGet() 诊断信息的错误句柄。
number1（输入）	被乘数。
number2（输入）	乘数。
result（输出）	积，需要提前使用NCIDescriptorAlloc分配空间。

- 示例

```
NCINumberMul(error, number_input, number_input2, result);
```

### 3.11.13. NCINumbersub

- 功能

两个NCINumber类型相减。

- 函数

```
sword NCINumberSub(  
    NCLError *err,  
    const NCINumber *number1,  
    const NCINumber *number2,  
    NCINumber *result)
```

- 参数

表 3.97. NCINumbersub 参数说明

名称	描述
err（输入/输出）	出现错误时可以传递给NCLErrorGet() 诊断信息的错误句柄。
number1（输入）	减数。
number2（输入）	被减数。
result（输出）	结果。

- 示例

```
NCINumberMod(error, number_int, number_int2, result);
```

## 3.12. 直接文件操作

### 3.12.1. NCIDirPathPrepare

- 功能

准备直接文件操作。

- 函数

```
sword NCIDirPathPrepare (  
    NCIDirPathCtx *dpctx,  
    NCISvcCtx *svchp,  
    NCIError *errhp)
```

- 参数

表 3.98. NCIDirPathPrepare 参数说明

名称	描述
dpctx (输入)	直接文件操作句柄。
svchp (输入)	执行所用的上下文句柄。
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。

在执行文件函数之前，文件操作的信息都要在文件操作环境的上下文属性中预先设置好。

- 注意

在调用该函数之前，先要在dpctx 上下文件中设置好要操作的表信息如NCI\_ATTR\_SUB\_NAME和NCI\_ATTR\_SCHEMA\_NAME、NCI\_ATTR\_NAME、NCI\_ATTR\_DATA\_TYPE。

- 示例

```
NCIDirPathPrepare(dpctx, sc, error);
```

### 3.12.2. NCIDirPathColArrayEntrySet

- 功能

设置写入文件数据数组上指定位置的数据内容。

- 函数

```
sword NCIDirPathColArrayEntrySet(  
    NCIDirPathColArray *dpca,  
    NCIError *errhp,  
    ub4 rownum,  
    ub2 colIdx,  
    ub1 *cvalp,
```

ub4 clen,  
ub1 cflg )

- 参数

表 3.99. NCIDirPathColArrayEntrySet 参数说明

名称	描述
dpca (输入)	直接文件操作句柄。
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
rownum (输入)	设置的数据内容在数组中的行号。
colIdx (输入)	设置的数据内容在数组中的列号。
cvalp (输入)	设置的数据内容缓冲区指针。
clen (输入)	数据内容的长度。
cflg (输入)	数据属性标识，如下所示。 <ul style="list-style-type: none"> <li>• NCI_DIRPATH_COL_COMPLETE: 在分批设置数据时，使用该标识表示数据已经结束。</li> <li>• NCI_DIRPATH_COL_NULL: 表示设置的数据是一个空(NULL)。</li> <li>• NCI_DIRPATH_COL_PARTIAL: 分批设置数据，表示当前设置的数据只是一部分。</li> </ul>

- 注意

当前行号、列号仅支持按顺序递增，如rownum 0, colIdx 0; rownum 0, colIdx 1; rownum 1, colIdx 0; rownum 1, colIdx 1。暂不支持倒序或其他方式输入。

- 示例

```
NCIDirPathColArrayEntrySet(dpca, error, rowoff, coloff, (ub1*)number_int, length,
NCI_DIRPATH_COL_COMPLETE);
```

### 3.12.3. NCIDirPathColArrayToStream

- 功能

把数据数组转换成为数据流。

- 函数

```
sword NCIDirPathColArrayToStream(
NCIDirPathColArray *dpca,
NCIDirPathCtx const *dpctx,
NCIDirPathStream *dpstr,
NCIError *errhp,
ub4 rowcnt,
ub4 rowoff)
```

- 参数

表 3.100. NCIDirPathColArrayToStream 参数说明

名称	描述
dpca (输入)	直接路径列数组句柄。
dpctx (输入)	直接文件操作句柄。
dpstr (输入)	数据流描述句柄。
errhp (输入/输出)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。
rowcnt (输入)	要转换的行数，保留参数。
rowoff (输入)	转换时在原数组上的起始偏移行号，保留参数。

- 示例

```
NCIDirPathColArrayToStream(dpca, dpctx, dpstr, error, 0, 0);
```

### 3.12.4. NCIDirPathLoadStream

- 功能

写入转换后的数据流到文件。

- 函数

```
sword NCIDirPathLoadStream (
    NCIDirPathCtx *dpctx,
    NCIDirPathStream *dpstr,
    NCIError *errhp )
```

- 参数

表 3.101. NCIDirPathLoadStream 参数说明

名称	描述
dpctx (输入)	直接文件操作句柄。
dpstr (输入/输出)	数据流描述句柄。
errhp (输入)	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。

- 示例

```
NCIDirPathLoadStream(dpctx, dpstr, error);
```

### 3.12.5. NCIDirPathFinish

- 功能

完成直接文件操作，清空相应的环境。

- 函数



```
sword NCIDirPathFinish (  
    NCIDirPathCtx *dpctx,  
    NCIError *errhp )
```

- 参数

表 3.102. NCIDirPathFinish 参数说明

名称	描述
dpctx（输入）	直接文件操作句柄。
errhp（输入/输出）	出现错误时可以传递给NCIErrorGet() 诊断信息的错误句柄。

- 示例

```
NCIDirPathFinish(dpctx, error);
```

## 3.13. 字符串操作

### 3.13.1. NCIStringAllocSize

- 功能

获得字符串所占内存空间的大小。

- 函数

```
sword NCIStringAllocSize(  
    NCIEnv *env,  
    NCIError *err,  
    CONST NCIString *vs,  
    ub4 *allocsize )
```

- 参数

表 3.103. NCIStringAllocSize 参数说明

名称	描述
env（输入/输出）	已初始化的环境句柄。
err（输入/输出）	错误信息句柄。
vs（输入）	需要获得字节数的字符串，不能为空。
allocsize（输出）	字符串所占内存大小(字节数)。

- 示例

```
NCIStringAllocSize(env, error, vs, &allocsize);
```

### 3.13.2. NCIStringAssign

- 功能

将一个字符串赋值给另一个字符串。

- 函数

```
sword NCIStrAssign (
    NCIEnv *env,
    NCLError *err,
    CONST NCIStr *rhs,
    NCIStr **lhs )
```

- 参数

表 3.104. NCIStrAssign 参数说明

名称	描述
env (输入/输出)	已初始化的环境句柄。
err (输入/输出)	错误信息句柄。
rhs (输入)	源字符串。
lhs (输入/输出)	目的字符串。

- 注意

该函数用于赋值字符串 rhs 给 lhs。字符串 lhs 的大小由 rhs 决定，字符串以 '\0' 为结束符。整个字符串的长度不包括结束符所占的字节数。

- 示例

```
NCIStrAssign(env, error, rhs, &lhs);
```

### 3.13.3. NCIStrAssignText

- 功能

将源文本串赋值给目的字符串。

- 函数

```
sword NCIStrAssignText (
    NCIEnv *env,
    NCLError *err,
    CONST OraText *rhs,
    ub2 rhs_len,
    NCIStr **lhs )
```

- 参数

表 3.105. NCIStrAssignText 参数说明

名称	描述
env (输入/输出)	已初始化的环境句柄。
err (输入/输出)	错误信息句柄。
rhs (输入)	源字符串。

名称	描述
rhs_len(输入)	rhs 字符串的长度。
lhs (输入/输出)	目的字符串。

- 注意

该函数用于赋值 rhs 给 lhs。字符串 lhs 的大小由 rhs 决定，如果rhs\_len的长度大于rhs的长度，以rhs的长度决定rhs的大小。分配的字符串以' \0' 为结束符。整个字符串的长度不包括结束符所占的字节数。

- 示例

```
NCIStringAssignText(env, error, rhs, rhs_len, &vs);
```

### 3.13.4. NCIStringPtr

- 功能

获得指向所给字符串文本的指针。

- 函数

```
Text* NCIStringPtr (  
    NCIEnv *env,  
    CONST NCIString *vs )
```

- 参数

表 3.106. NCIStringPtr 参数说明

名称	描述
env (输入/输出)	已初始化的环境句柄。
vs (输入)	返回指向 NCIString 对象的指针。

- 示例

```
NCIStringPtr(env, vs);
```

### 3.13.5. NCIStringResize

- 功能

为给定的字符串重新指定所占内存的大小。

- 函数

```
sword NCIStringResize (  
    NCIEnv *env,  
    NCIError *err,  
    ub4 new_size,  
    NCIString **str )
```

- 参数

表 3.107. NCIStrResize 参数说明

名称	描述
env (输入/输出)	已初始化的环境句柄。
err (输入/输出)	错误信息句柄。
new_size (输入)	调整后字符串所占内存的字节数。
str (输入/输出)	字符串需要申请的空间，其初始内存将被 NCI 对象缓存释放。

- 注意

该函数用于调整对象缓存中可变长度的字符串的内存大小，字符串的内容不被保存。函数可给字符串分配一个新的内存空间，这时源内存空间将被释放。如果 str 为空，函数将直接给字符串分配空间。如果 new\_size 为 0，函数会释放字符串 str 所占的内存，并返回一个空指针值。

- 示例

```
NCIStrResize(env, error, new_size, &str);
```

### 3.13.6. NCIStrSize

- 功能

获得字符串的长度。

- 函数

```
sword NCIStrSize (  
    NCIEnv *env,  
    CONST NCIStr *vs )
```

- 参数

表 3.108. NCIStrSize 参数说明

名称	描述
env (输入/输出)	已初始化的环境句柄。
vs (输出)	需要处理的字符串。

- 注意

返回的字符串长度不包括 ‘\0’ 结束符所占的字节。

- 示例

```
NCIStrSize(env, vs);
```

---

# 第 4 章 附录

## 4.1. 使用示例

1. 由于显式的调用SQL语句并不会被NCI感知到，这可能导致NCI本身对事务的管理和大对象的管理出现错误，请避免显式的调用以下语句。
  - a. 事务相关操作。
  - b. 服务端的大对象操作接口。
2. 使用大对象接口之前需要创建ux\_lob\_operate插件。

### 4.1.1. 编译命令

```
gcc -o test test.c -DNCI_EXPORT -I/home/uxdb/uxdbinstall/dbsql/include/ -L/home/uxdb/uxdbinstall/dbsql/lib/nci -lncli -Wl,-rpath=/home/uxdb/uxdbinstall/dbsql/lib/nci
```

### 4.1.2. 创建表

```
#include <string.h>
#include <nci.h>

int main(int argc, char **argv)
{
    int id = 0;
    int val = 0;
    NCIDefine *bhp1 = NULL;
    NCIDefine *bhp2 = NULL;
    ub2 datalen = 0;
    sb4 rows_fetched = 0;
    sb4 cols_fetched = 0;
    ub4 sqllen = 0;
    char *dbname = "localhost:5432/uxdb";
    char *user = "uxdb";
    char *password = "lqaz!QAZ";
    /* 初始化环境句柄 */
    NCIEnv *envhpp = NULL;
    /* 初始化服务句柄 */
    NCIServer *servhpp = NULL;
    /* 初始化捕获错误句柄 */
    NCIErr *errhpp = NULL;
    /* 初始化会话句柄 */
    NCISession *usrhpp = NULL;
    /* 初始化服务上下文句柄 */

    NCISvcCtx *svchpp = NULL;
    /* 初始化操作句柄 */
    NCISmt *stmthpp = NULL;
    /* 创建NCI环境 */
```

```

sword swResult = NCIEncCreate(&envhpp, NCI_DEFAULT, NULL, NULL, NULL, NULL, 0,
NULL);
if(swResult != NCI_SUCCESS && swResult != NCI_SUCCESS_WITH_INFO)
{
printf("NCIEncCreate Error!\n");
return -1;
}
/* 创建错误句柄 */
NCIHandleAlloc((dvoid *)envhpp, (dvoid **)&errhpp, NCI_HTYPE_ERROR, (size_t)0, (dvoid
***)0);
printf("&errhpp = %p\n", errhpp);
/* 创建服务句柄 */
NCIHandleAlloc((dvoid *)envhpp, (dvoid **)&servhpp, NCI_HTYPE_SERVER, (size_t)0, (dvoid
***)0);
printf("&servhpp = %p\n", servhpp);
/* 连接服务器 */
if (NCIServerAttach(servhpp, errhpp, (text *)dbname, strlen(dbname), NCI_DEFAULT) !=
NCI_SUCCESS)
{
printf("NCIServerAttach Error!\n");
return -1;
}
/* 创建服务上下文句柄 */

(void)NCIHandleAlloc((dvoid *)envhpp, (dvoid **)&svchpp, NCI_HTYPE_SVCCTX, (size_t)0,
(dvoid ***)0);
printf("&svchpp = %p\n", svchpp);

/* 设置属性 */
(void)NCIAttrSet((dvoid *)svchpp, NCI_HTYPE_SVCCTX, (dvoid *)servhpp, (ub4)0,
NCI_ATTR_SERVER, (NCIError *)errhpp);
/* 创建用户连接句柄 */
(void)NCIHandleAlloc((dvoid *)envhpp, (dvoid **)&usrhpp, (ub4)NCI_HTYPE_SESSION,
(size_t)0, (dvoid ***)0);
printf("&usrhpp = %p\n", usrhpp);
/* 设置用户名 */
(void)NCIAttrSet((dvoid *)usrhpp, (ub4)NCI_HTYPE_SESSION, (dvoid *)user, (ub4)strlen(user),
(ub4)NCI_ATTR_USERNAME, errhpp);
/* 设置密码 */
(void)NCIAttrSet((dvoid *)usrhpp, (ub4)NCI_HTYPE_SESSION, (dvoid *)password,
(ub4)strlen(password), (ub4)NCI_ATTR_PASSWORD, errhpp);
/* 开启连接 */
if(NCISessionBegin(svchpp, errhpp, usrhpp, NCI_CRED_RDBMS, (ub4)NCI_DEFAULT) !=
NCI_SUCCESS)
{
NCIText errbuf[1024] = {0};
sb4 errcode = 0;
NCIText sqlstate[1024];
NCIErrorGet((dvoid *)errhpp, (ub4)1, sqlstate, &errcode, (NCIText *)errbuf, (ub4)sizeof(errbuf),
NCI_HTYPE_ERROR);
printf("Error:%s\n", errbuf);
return -1;
}

```

```

}
/* 创建操作语句句柄 */
NCIHandleAlloc((dvoid *)envhpp, (dvoid **)&stmthpp, NCI_HTYPE_STMT, (size_t)0, (dvoid
***)0);
printf("&stmthpp = %p\n", stmthpp);

/* 定义sql语句 */
NCIText sql1[255] = "create table tb1(id int, val int, val2 int);";
NCIText sql2[255] = "insert into tb1 values(1,10,33);";
NCIText sql3[255] = "insert into tb1 values(11,100,333);";
NCIText sql4[255] = "insert into tb1 values(111,1000,3333);";
NCIText sql5[255] = "insert into tb1 values(1111,10000,33333);";
NCIText sql6[255] = "select id,val from tb1 where id < 222;";
sb4 ret = 0;
NCIText err[128] = {"\0"};
if (NCISmtPrepare(stmthpp, errhpp, (text *)sql1, strlen(sql1), (ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT) != NCI_SUCCESS)
{
printf("[ERROR] NCISmtPrepare1\n");
return -1;
}
ret = NCISmtExecute(svchpp, stmthpp, errhpp, (ub4)0, (ub4)0, NULL, NULL, NCI_DEFAULT);
if(NCI_SUCCESS != ret)
{
NCIErrorGet(envhpp, 0, (NCIText *)0, (sb4)0, (NCIText *)err, sizeof(err),
NCI_HTYPE_ERROR);
printf("%s\n", err);
return -1;
}
if (NCISmtPrepare(stmthpp, errhpp, (text *)sql2, strlen(sql2), (ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT) != NCI_SUCCESS)
{
printf("[ERROR] NCISmtPrepare2\n");
return -1;
}
ret = NCISmtExecute(svchpp, stmthpp, errhpp, (ub4)0, (ub4)0, NULL, NULL, NCI_DEFAULT);

if(NCI_SUCCESS != ret)
{
NCIErrorGet(envhpp, 0, (NCIText *)0, (sb4)0, (NCIText *)err, sizeof(err),
NCI_HTYPE_ERROR);
printf("%s\n", err);
return -1;
}
if (NCISmtPrepare(stmthpp, errhpp, (text *)sql3, strlen(sql3), (ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT) != NCI_SUCCESS)
{
printf("[ERROR] NCISmtPrepare3\n");
return -1;
}
ret = NCISmtExecute(svchpp, stmthpp, errhpp, (ub4)0, (ub4)0, NULL, NULL, NCI_DEFAULT);
if(NCI_SUCCESS != ret)

```

```

{
    NCLErrorGet(envhpp, 0, (NCIText *)0, (sb4)0, (NCIText *)err, sizeof(err),
    NCI_HTYPE_ERROR);
    printf("%s\n", err);
    return -1;
}
if (NCIStmtPrepare(stmthpp, errhpp, (text *)sql4, strlen(sql4), (ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT) != NCI_SUCCESS)
{
    printf("[ERROR] NCIStmtPrepare4\n");
    return -1;
}
ret = NCIStmtExecute(svchpp, stmthpp, errhpp, (ub4)0, (ub4)0, NULL, NULL, NCI_DEFAULT);
if(NCI_SUCCESS != ret)
{

    NCLErrorGet(envhpp, 0, (NCIText *)0, (sb4)0, (NCIText *)err, sizeof(err),
    NCI_HTYPE_ERROR);
    printf("%s\n", err);
    return -1;
}
if (NCIStmtPrepare(stmthpp, errhpp, (text *)sql5, strlen(sql5), (ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT) != NCI_SUCCESS)
{
    printf("[ERROR] NCIStmtPrepare4\n");
    return -1;
}
ret = NCIStmtExecute(svchpp, stmthpp, errhpp, (ub4)0, (ub4)0, NULL, NULL, NCI_DEFAULT);
if(NCI_SUCCESS != ret)
{
    NCLErrorGet(envhpp, 0, (NCIText *)0, (sb4)0, (NCIText *)err, sizeof(err),
    NCI_HTYPE_ERROR);
    printf("%s\n", err);
}
/* 准备sql语句 */
//sqllen = NCIStrSize(envhpp, sql6);
sqllen = strlen(sql6);
if (NCIStmtPrepare(stmthpp, errhpp, (text *)sql6, sqllen, (ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT) != NCI_SUCCESS)
{
    printf("[ERROR] NCIStmtPrepare6\n");
    return -1;
}
NCIDefineByPos(stmthpp, &bhp1, errhpp, 1, (dvoid *) &id, sizeof(id), SOLT_INT, NULL,
&datalen, NULL, NCI_DEFAULT);
NCIDefineByPos(stmthpp, &bhp2, errhpp, 2, (dvoid *) &val, sizeof(val), SOLT_INT, NULL,
&datalen, NULL, NCI_DEFAULT);

/* 执行sql语句 */
NCIStmtExecute(svchpp, stmthpp, errhpp, (ub4)0, (ub4)0, NULL, NULL, NCI_DEFAULT);

```



```

do
{
printf("id = %d, val = %d\n", id, val);
} while (NCISstmtFetch(stmthpp, errhpp, 1, NCI_FETCH_NEXT, NCI_DEFAULT) !=
NCI_NO_DATA);
/* 获取执行结果 */
ret = NCIAAttrGet((CONST void *)stmthpp, NCI_HTYPE_STMT, (void *) &rows_fetched, (ub4
*)sizeof(rows_fetched), NCI_ATTR_ROW_COUNT, errhpp);
printf("rows = %d,ret = %d\n", rows_fetched, ret);

/* 断开服务器连接 */
NCIServerDetach(servhpp, errhpp, NCI_DEFAULT);
/* 释放内存资源 */
ret = NCIHandleFree((dvoid *)stmthpp, NCI_HTYPE_STMT);
printf("ret = %d\n", ret);
ret = NCIHandleFree((dvoid *)svchpp, NCI_HTYPE_SVCCTX);
printf("ret = %d\n", ret);
ret = NCIHandleFree((dvoid *)servhpp, NCI_HTYPE_SERVER);
printf("ret = %d\n", ret);
ret = NCIHandleFree((dvoid *)errhpp, NCI_HTYPE_ERROR);
printf("ret = %d\n", ret);
ret = NCIHandleFree((dvoid *)usrhpp, NCI_HTYPE_SESSION);
printf("ret = %d\n", ret);

return 0;
}

```

### 4.1.3. 插入数据及事务提交

```

#include <string.h>
#include <unistd.h>
#include "nci.h"

int main(int argc, char **argv)
{
int id = 0;
int val = 0;
NCIDefine *bhp1 = NULL;
NCIDefine *bhp2 = NULL;
ub2 datalen = 0;
int rows_fetched = 0;
char *dbname = "localhost:5433/uxdb";
char *user = "uxdb";
char *password = "1qaz!QAZ";
sb4 errcode;
NCIText sqlstate[128];

/* 初始化环境句柄 */

```

```
NCIEnv *envhpp = NULL;
/* 初始化服务句柄 */
NCIServer *servhpp = NULL;
/* 初始化捕获错误句柄 */
NCIError *errhpp = NULL;
/* 初始化会话句柄 */
NCISession *usrhpp = NULL;

/* 初始化服务上下文句柄 */
NCISvcCtx *svchpp = NULL;
/* 初始化操作句柄 */
NCISmt *stmthpp = NULL;

/* 创建NCI环境 */
sword swResult = NCIEnvCreate(&envhpp, NCI_DEFAULT, NULL, NULL, NULL, NULL, 0, NULL);
if(swResult != NCI_SUCCESS && swResult != NCI_SUCCESS_WITH_INFO)
{
    printf("NCIEnvCreate Error!\n");
    return -1;
}
/* 创建错误句柄 */
NCIHandleAlloc((dvoid *)envhpp, (dvoid **)&errhpp, NCI_HTYPE_ERROR, (size_t)0, (dvoid **)&0);
/* 创建服务句柄 */
NCIHandleAlloc((dvoid *)envhpp, (dvoid **)&servhpp, NCI_HTYPE_SERVER, (size_t)0, (dvoid
**&0);
/* 连接服务器 */
if (NCIServerAttach(servhpp, errhpp, (text *)dbname, strlen(dbname), NCI_DEFAULT) !=
NCI_SUCCESS)
{
    printf("NCIServerAttach Error!\n");
    return -1;
}
/* 创建错误句柄 */
(void)NCIHandleAlloc((dvoid *)envhpp, (dvoid **)&errhpp, NCI_HTYPE_ERROR, (size_t)0, (dvoid
**&0);
/* 创建服务上下文句柄 */
(void)NCIHandleAlloc((dvoid *)envhpp, (dvoid **)&svchpp, NCI_HTYPE_SVCCTX, (size_t)0,
(dvoid **)&0);
/* 设置属性 */
(void)NCIAttrSet((dvoid *)svchpp, NCI_HTYPE_SVCCTX, (dvoid *)servhpp, (ub4)0,
NCI_ATTR_SERVER,
(NCIError *)errhpp);
/* 创建用户连接句柄 */
(void)NCIHandleAlloc((dvoid *)envhpp, (dvoid **)&usrhpp, (ub4)NCI_HTYPE_SESSION, (size_t)0,
(dvoid **)&0);
/* 设置用户名 */
(void)NCIAttrSet((dvoid *)usrhpp, (ub4)NCI_HTYPE_SESSION, (dvoid *)user, (ub4)strlen(user),
(ub4)NCI_ATTR_USERNAME, errhpp);
/* 设置密码 */
(void)NCIAttrSet((dvoid *)usrhpp, (ub4)NCI_HTYPE_SESSION, (dvoid *)password,
(ub4)strlen(password),
(ub4)NCI_ATTR_PASSWORD, errhpp);
/* 开启连接 */
```

---

```

if(NCISessionBegin(svchpp, errhpp, usrhpp, NCI_CRED_RDBMS, (ub4)NCI_DEFAULT) !=
NCI_SUCCESS)
{
char errbuf[128] = {'\0'};
NCIErrorGet((dvoid *)errhpp, (ub4)1, (NCIText *)sqlstate, &errcode, (NCIText *)errbuf,
(ub4)sizeof(errbuf), NCI_HTYPE_ERROR);
printf("Error:%s\n", errbuf);
return -1;
}
/* 创建操作语句句柄 */
NCIHandleAlloc((dvoid *)envhpp, (dvoid **)&stmthpp, NCI_HTYPE_STMT, (size_t)0, (dvoid **)&0);
/* 定义sql语句 */
char sqlcreate[255] = "create table IF NOT EXISTS tb1(id int, val int)";
char sql1[255] = "insert into tb1 values(6, 666)";
char sql2[255] = "insert into tb1 values(66, 6666)";
char sql3[255] = "select id,val from tb1";
char sql4[255] = "update tb1 set val = 7777 where id = 66";
int ret;
char err[128] = {'\0'};
/*
* case1:正常执行事务提交成功
* 执行一个insert语句, 事务正常提交成功 */
NCITransStart(svchpp, errhpp, 60, NCI_TRANS_NEW);

if (NCISmtPrepare(stmthpp, errhpp, (text *)sqlcreate, strlen(sqlcreate),
(ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT) != NCI_SUCCESS)
{
printf("[ERROR] NCISmtPrepare1\n");
return -1;
}
ret = NCISmtExecute(svchpp, stmthpp, errhpp, (ub4)0, (ub4)0, NULL, NULL, NCI_DEFAULT);
if(NCI_SUCCESS != ret)
{
NCIErrorGet(envhpp, 0, (NCIText *)sqlstate, &errcode, (NCIText *)err, sizeof(err),
NCI_HTYPE_ERROR);
printf("[ERROR] sqlcreate %s\n", err);
return -1;
}

if (NCISmtPrepare(stmthpp, errhpp, (text *)sql1, strlen(sql1), (ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT) != NCI_SUCCESS)
{
printf("[ERROR] NCISmtPrepare1\n");
return -1;
}
ret = NCISmtExecute(svchpp, stmthpp, errhpp, (ub4)0, (ub4)0, NULL, NULL, NCI_DEFAULT);
if(NCI_SUCCESS != ret)
{
NCIErrorGet(envhpp, 0, (NCIText *)sqlstate, &errcode, (NCIText *)err, sizeof(err),
NCI_HTYPE_ERROR);
printf("[ERROR] NCISmtExecute1 %s\n", err);
}

```

---

---

```

return -1;
}
if (NCISmtPrepare(stmthpp, errhpp, (text *)sql2, strlen(sql2), (ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT) != NCI_SUCCESS)
{
printf("[ERROR] NCISmtPrepare2\n");
return -1;
}
ret = NCISmtExecute(svchpp, stmthpp, errhpp, (ub4)0, (ub4)0, NULL, NULL, NCI_DEFAULT);
if(NCI_SUCCESS != ret)
{
NCIErrorGet(envhpp, 0, (NCIText *)sqlstate, &errcode, (NCIText *)err, sizeof(err),
NCI_HTYPE_ERROR);
printf("[ERROR] NCISmtExecute2 %s\n", err);
return -1;
}
if (NCISmtPrepare(stmthpp, errhpp, (text *)sql4, strlen(sql4), (ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT) != NCI_SUCCESS)
{
printf("[ERROR] NCISmtPrepare3\n");
return -1;
}
ret = NCISmtExecute(svchpp, stmthpp, errhpp, (ub4)0, (ub4)0, NULL, NULL, NCI_DEFAULT);
if(NCI_SUCCESS != ret)
{
NCIErrorGet(envhpp, 0, (NCIText *)sqlstate, &errcode, (NCIText *)err, sizeof(err),
NCI_HTYPE_ERROR);
printf("[ERROR] NCISmtExecute3 %s\n", err);
return -1;
}
ret = NCITransCommit(svchpp, errhpp, 0);
if (ret != NCI_SUCCESS)
{
NCIErrorGet(envhpp, 0, (NCIText *)sqlstate, &errcode, (NCIText *)err, sizeof(err),
NCI_HTYPE_ERROR);
printf("NCITransCommit: case 1 test error!\n");
printf("result=%d,errmsg=%s\n", ret, err);
}
else
printf("NCITransCommit: case 1 test success!\n");
/*
* case2:构造异常情况测试返回值和error信息
* 1) 构造服务器连接异常错误
* 2) 构造重复提交错误
*/
/*此处停止数据库, 则进行异常情况测试, 可以打印错误信息 */
/* 不停止数据库, 则继续正常处理 */
printf("If stop db to generated an error!\n");
sleep(2);
ret = NCITransCommit(svchpp, errhpp, 0);
if (ret != NCI_SUCCESS)
{
NCIErrorGet(envhpp, 0, (NCIText *)sqlstate, &errcode, (NCIText *)err, sizeof(err),
NCI_HTYPE_ERROR);

```

---

```

printf("NCITransCommit: case 2 test error result!\n");
printf("result=%d,errmsg=%s\n", ret, err);
}
else
printf("NCITransCommit: case 2 test sucess result!\n");
ub4 sqllen = strlen(sql3);
if (NCISmtPrepare(stmthpp, errhpp, (text *)sql3, sqllen, (ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT) != NCI_SUCCESS)
{
printf("[ERROR] NCISmtPrepare6\n");
return -1;
}
NCIDefineByPos(stmthpp, &bhp1, errhpp, 1, (dvoid *)&id, sizeof(id), SOLT_INT, NULL, &datalen,
NULL, NCI_DEFAULT);
NCIDefineByPos(stmthpp, &bhp2, errhpp, 2, (dvoid *)&val, sizeof(val), SOLT_INT, NULL, &datalen,
NULL, NCI_DEFAULT);
/* 执行sql语句 */
NCISmtExecute(svchpp, stmthpp, errhpp, (ub4)0, (ub4)0, NULL, NULL, NCI_DEFAULT);
while (NCISmtFetch(stmthpp, errhpp, 1, NCI_FETCH_NEXT, NCI_DEFAULT) != NCI_NO_DATA)
{
printf("id = %d, val = %d\n", id, val);
}
/* 获取执行结果 */
NCIAttrGet((CONST void *)stmthpp, NCI_HTYPE_STMT, (void *)&rows_fetched,
(ub4 *)sizeof(rows_fetched), NCI_ATTR_ROW_COUNT, errhpp);
printf("rows:%d\n", rows_fetched);
/* 断开服务器连接 */
NCIServerDetach(servhpp, errhpp, NCI_DEFAULT);
/*释放内存资源 */
NCIHandleFree((dvoid *)stmthpp, NCI_HTYPE_STMT);
NCIHandleFree((dvoid *)svchpp, NCI_HTYPE_SVCCTX);
NCIHandleFree((dvoid *)servhpp, NCI_HTYPE_SERVER);
NCIHandleFree((dvoid *)errhpp, NCI_HTYPE_ERROR);
return 0;
}

```

#### 4.1.4. 多行绑定

```

#include <string.h>
#include <nci.h>

#define ARRAYSIZE 10
#define NAME_LEN 20
typedef struct
{
    char id[10];
    char sname[20];
    int age;
    char sex[10];
}stuData;
/* 指示器数组 */
typedef struct

```

---

```

{
    sb2 sb2_id[ARRAYSIZE];
    sb2 sb2_sname[ARRAYSIZE];
    sb2 sb2_age[ARRAYSIZE];
    sb2 sb2_sex[ARRAYSIZE];
} stdInd_T;

/* 字段长度数组 */
typedef struct
{
    ub2 ub2_id[ARRAYSIZE];
    ub2 ub2_sname[ARRAYSIZE];
    ub2 ub2_age[ARRAYSIZE];
    ub2 ub2_sex[ARRAYSIZE];
} stdLen_T;

stuData tstd[ARRAYSIZE]; /* 数组变量，用于批量操作 */
NCIBind *bindp[ARRAYSIZE] = {0};
NCIDefine *defnp[ARRAYSIZE] = {0};
stuData tstd[ARRAYSIZE]; /* 数组变量，用于批量操作 */
stdInd_T tstdInd;
stdLen_T tstdLen;
stdLen_T tstdRet;
stuData results[ARRAYSIZE];

sb2 sb2aIndid[ARRAYSIZE] = {0}; /* 指示器变量，用于取可能存在空值的字 */
ub2 datalen[ARRAYSIZE] = {0}; /* 获取数据长度 */
char id[NAME_LEN] = {0};
char sname[NAME_LEN] = {0};
int age = 0;
char sex[NAME_LEN] = {0};

static void init_bind_parameter()
{
    int i = 0;
    memset(tstd, 0, ARRAYSIZE * sizeof(stuData));
    for (i = 0; i < ARRAYSIZE; i++)
    {
        snprintf(tstd[i].id, 10, "ID%d", i);
        snprintf(tstd[i].sname, 20, "李%d", i);
        tstd[i].age = i + 10;
        if(i % 2 == 0)
            snprintf(tstd[i].sex, 10, "%s", "女");
        else
            snprintf(tstd[i].sex, 10, "%s", "男");
    }
}

CONST OraText dbname[] = "localhost:5432"; /* ip:port/dbname */
OraText username[100] = "uxdb";
CONST OraText pwd[] = "1qaz!QAZ";

int main(int argc, char **argv)

```

---

```

{
/* 初始化环境句柄 */
NCIEnv *envhpp = NULL;
/* 初始化服务句柄 */
NCIServer *servhpp = NULL;
/* 初始化捕获错误句柄 */
NCIError *errhpp = NULL;
/* 初始化会话句柄 */
NCISession *usrhpp = NULL;
/* 初始化服务上下文句柄 */

NCISvcCtx *svchpp = NULL;
/* 初始化操作句柄 */
NCISmt *stmthpp = NULL;
NCIEnv *env = NULL;
NCISvcCtx *sc = NULL;
NCISmt *stmt = NULL;
NCIError *error = NULL;
int i = 0;
int rowoff = 5;
int update_or_delete_rownum = 2;
int rowoff1 = 1;
int new_age[2] = {20,21};
char pid[2][10] = {0};
char sqlselect[] = "select id,sname,age,sex from stu";
char sqldrop[] = "drop table if exists stu;";
char sqlcreate[] = "create table stu(id varchar(100) not null , sname varchar, age int, sex varchar); ";
char sqlbind[] = "insert into stu(id,sname,age,sex) values(:Vhid,:Vhname,:Vhage,:Vhsex)";
int retcode = -1;

    retcode = NCIInitialize((ub4)NCI_DEFAULT, (dvoid *)0, (dvoid * (*)(dvoid *, size_t))0, (dvoid * (*)(dvoid *, dvoid *, size_t))0, (void (*)(dvoid *, dvoid *))0);
    if (NCI_SUCCESS != retcode)
    {
        printf("error : NCIInitialize!\n");
        return retcode;
    }
    retcode = NCIEnvInit((dvoid *)&env, (ub4)NCI_DEFAULT, (size_t)0, (dvoid **)0);
    if (NCI_SUCCESS != retcode)
    {
        printf("error : NCIEnvInit!\n");
        return retcode;
    }
    retcode = NCIHandleAlloc(env, (dvoid *)&sc, (ub4)NCI_HTYPE_SVCCTX, (size_t)0, (dvoid **)0);
    if (NCI_SUCCESS != retcode)
    {
        printf("I failed to allocate connection handle!\n");
        return retcode;
    }
    retcode = NCIHandleAlloc(env, (dvoid *)&error, (ub4)NCI_HTYPE_ERROR, (size_t)0, (dvoid **)0);
    if (NCI_SUCCESS != retcode)
    {

```

```

    printf("2failed to allocate error handle!\n");
    return retcode;
}
retcode = NCILogon(env, error, &sc, username, strlen((char *)username), pwd, strlen((char *)pwd),
dbname, strlen((char *)dbname));
if (NCI_SUCCESS != retcode)
{
    printf("unable to connect to the database!\n");
    return retcode;
}
retcode = NCIHandleAlloc(env, (dvoid *)&stmt, (ub4)NCI_HTYPE_STMT, (size_t)0,
(dvoid **)0);
if (NCI_SUCCESS != retcode)
{
    printf("failed to allocate statement handle!\n");
    return retcode;
}
/* 删除表 */
retcode = NCISmtPrepare(stmt, error, (text *)sqldrop, (ub4)strlen(sqldrop),
(ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT);
if (NCI_SUCCESS != retcode)
{
    printf("failed to prepare sql!\n");
    return retcode;
}

retcode = NCISmtExecute(sc, stmt, error, (ub4)1, (ub4)0, (NCISnapshot *)NULL, (NCISnapshot
*)NULL, (ub4)NCI_DEFAULT);
if (NCI_SUCCESS != retcode)
{
    printf("failed to execute sql!\n");
    return retcode;
}
/* 创建表 */
retcode = NCISmtPrepare(stmt, error, (text *)sqlcreate, (ub4)strlen(sqlcreate),
(ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT);
if (NCI_SUCCESS != retcode)
{
    printf("failed to prepare sql!\n");
    return retcode;
}

retcode = NCISmtExecute(sc, stmt, error, (ub4)1, (ub4)0, (NCISnapshot *)NULL, (NCISnapshot
*)NULL, (ub4)NCI_DEFAULT);
if (NCI_SUCCESS != retcode)
{
    printf("failed to execute sql!\n");
    return retcode;
}
/* 初始化变量数组 */
init_bind_parameter();

```



```

/*****/
/*****/ 参数绑定 *****/
/*****/
retcode = NCISmtPrepare(stmt, error, (text *)sqlbind, (ub4)strlen(sqlbind),
                        (ub4)NCI_NTV_SYNTAX, (ub4)NCI_DEFAULT);

retcode = NCIBindByPos(stmt, &bindp[0], error, 1, tstd[0].id, sizeof(tstd[0].id), SQLT_STR,
&tstdInd.sb2_id[0], 0, 0, 0, 0, NCI_DEFAULT);

retcode = NCIBindByPos(stmt, &bindp[1], error, 2, tstd[0].sname, sizeof(tstd[0].sname), SQLT_STR,
&tstdInd.sb2_sname[0], 0, 0, 0, 0, NCI_DEFAULT);

retcode = NCIBindByPos(stmt, &bindp[2], error, 3, (dvoid *)&tstd[0].age,
sizeof(tstd[0].age),SQLT_INT, &tstdInd.sb2_age[0], (ub2 *)0, (ub2)0, (ub4)0, (ub4 *)0,
NCI_DEFAULT);

retcode = NCIBindByPos(stmt, &bindp[3], error, 4, tstd[0].sex, sizeof(tstd[0].sex), SQLT_STR,
&tstdInd.sb2_sex[0], (ub2 *)0, (ub2)0, (ub4)0, (ub4 *)0, NCI_DEFAULT);

/*****/
/*****/ 变量数组绑定 *****/
/*****/
retcode = NCIBindArrayOfStruct(bindp[0], error,sizeof(tstd[0]), 0, 0, 0);

retcode = NCIBindArrayOfStruct(bindp[1], error,sizeof(tstd[0]), 0, 0, 0);

retcode = NCIBindArrayOfStruct(bindp[2], error,sizeof(tstd[0]), 0, 0, 0);

retcode = NCIBindArrayOfStruct(bindp[3], error,sizeof(tstd[0]), 0, 0, 0);

/*****/
/*****/ 插入 *****/
/*****/
retcode = NCISmtExecute(sc, stmt, error, (ub4)ARRAYSIZE, (ub4)0, (NCISnapshot *)NULL,
(NCISnapshot *)NULL, (ub4)NCI_DEFAULT);

/*****/
/*****/ 查询 *****/
/*****/
retcode = NCISmtPrepare(stmt, error, (text *)sqlselect, (ub4)strlen(sqlselect),
                        (ub4)NCI_NTV_SYNTAX, (ub4)NCI_DEFAULT);

/* 绑定输出参数 */
retcode = NCIDefineByPos(stmt, &defnp[0], error, 1, (dvoid *)id, (ub4)sizeof(id),SQLT_STR,
&sb2aIndid[0], (ub2 *)&datalen[0], NULL, NCI_DEFAULT);

retcode = NCIDefineByPos(stmt, &defnp[1], error, 2, (dvoid *)sname,
(ub4)sizeof(sname),SQLT_STR, &sb2aIndid[1], (ub2 *)&datalen[1], NULL, NCI_DEFAULT);

retcode = NCIDefineByPos(stmt, &defnp[2], error, 3, (dvoid *)&age, (ub4)sizeof(age),SQLT_INT,
&sb2aIndid[2], (ub2 *)&datalen[2], NULL, NCI_DEFAULT);

retcode = NCIDefineByPos(stmt, &defnp[3], error, 4, (dvoid *)sex, (ub4)sizeof(sex),SQLT_STR,
&sb2aIndid[3], (ub2 *)&datalen[3], NULL, NCI_DEFAULT);

```

```

/* 执行SQL sqlselect */
retcode = NCISstmtExecute(sc, stmt, error, (ub4)0, (ub4)0, (NCISnapshot *)NULL, (NCISnapshot
*)NULL, (ub4)NCI_DEFAULT);

/*****
/*****      获取结果集      *****/
/*****
while ((retcode = NCISstmtFetch(stmt, error, 1, NCI_FETCH_NEXT, NCI_DEFAULT)) !=
NCI_NO_DATA)
{
    printf("id = %s, name = %s, age = %d, sex = %s\n", id, sname, age, sex);
}
return 0;
}

```

#### 4.1.5. 大对象

前提：需要用户找一个图片并命名为cat.gif。

```

#include <nci.h>

int retcode = 0;
/* 调用函数NCISstmtPrepare准备语句，调用函数NCISstmtExecute执行语句 */
sword nci_stmt_execute(NCISvcCtx *sc, NCISstmt *stmt, NCIErr *error, const char *sql)
{
    retcode = NCISstmtPrepare(stmt, error, (text *)sql, (ub4)strlen(sql), (ub4)NCI_NTV_SYNTAX,
(ub4)NCI_DEFAULT);
    if (NCI_SUCCESS != retcode)
    {
        printf("failed to prepare sql!\n");
        return retcode;
    }

    retcode = NCISstmtExecute(sc, stmt, error, (ub4)1, (ub4)0, (NCISnapshot *)NULL, (NCISnapshot
*)NULL, (ub4)NCI_DEFAULT);
    if (NCI_SUCCESS != retcode)
    {
        printf("failed to execute sql!\n");
        return retcode;
    }
}

/* 调用函数NCILogoff断开数据库连接，并释放环境、连接、语句、错误句柄 */
sword nci_logoff_disconnect(NCIEnv *env, NCISvcCtx *sc, NCISstmt *stmt, NCIErr *error)
{
    retcode = NCIHandleFree((dvoid *)stmt, (ub4)NCI_HTYPE_STMT);
    if (NCI_SUCCESS != retcode)
    {
        printf("failed to release statement handle!\n");
        return retcode;
    }

    retcode = NCILogoff(sc, error);
}

```

---

98

---

```

retcode = NCCHandleAlloc((dvoid *)env, (dvoid **)error, (ub4)NCI_HTYPE_ERROR, (size_t)0,
(dvoid **)0);
if (NCI_SUCCESS != retcode)
{
    printf("failed to allocate error handle!\n");
    return retcode;
}
retcode = NCILogon(*env, *error, sc, username, strlen((char *)username), password, strlen((char
*)password), dbname, strlen((char *)dbname));
if (NCI_SUCCESS != retcode)
{
    printf("unable to connect to the database!\n");
    return retcode;
}
retcode = NCCHandleAlloc((dvoid *)env, (dvoid **)stmt, (ub4)NCI_HTYPE_STMT, (size_t)0,
(dvoid **)0);
if (NCI_SUCCESS != retcode)
{
    printf("failed to allocate statement handle!\n");
    return retcode;
}

return retcode;
}
int main(void)
{
    NCIEnv *env = NULL;
    NCISvcCtx *svcctx = NULL;
    NCISmt *stmt = NULL;
    NCIErr *error = NULL;

    char usname[] = "UXDB";
    char pwd[] = "1qaz!QAZ";
    char dbname[] = "127.0.0.1:5432/UXDB";
    char bufferR[2048] = {0};
    char bufgifw[2048] = {0};
    const char *bufferW = "abcdefghijklmnopqrstuvwxyz";

    char sqldroplob[] = "drop table if exists blo;";
    char sqlcreatelob[] = "create table blo(id varchar,text blob, text2 clob);";
    char sqlinsertlob[] = "insert into blo(id,text,text2) values(:Vhid,:Vhtext,:Vhtext2)";
    char sqlselectlob[] = "select textout(id),text,text2 from blo where id = :id;";

    NCILobLocator *lob;
    NCILobLocator *lobgif;
    FILE *filepw, *filepr;

    ub2 indp = 1;
    ub4 amtp = strlen(bufferW);
    ub4 lob_offset = 0;
    ub1 piece = 0;
    ub4 size = 0;
/*
* 初始化

```

---

```

*/
nci_logon_connect(&env, &svcctx, &stmt, &error, username, pwd, dbname);

NCIDDescriptorAlloc((dvoid *)env, (dvoid **)&lob, (ub4)NCI_DTYPE_LOB, (size_t)0, (dvoid **)0);
NCILobCreateTemporary(svcctx, error, lob, 0, SQLCS_IMPLICIT, NCI_TEMP_BLOB,
NCI_ATTR_NOCACHE, (NCIDuration)0);
NCIDDescriptorAlloc((dvoid *)env, (dvoid **)&lobgif, (ub4)NCI_DTYPE_LOB, (size_t)0, (dvoid
**))0);
NCILobCreateTemporary(svcctx, error, lobgif, 0, SQLCS_IMPLICIT, NCI_TEMP_BLOB,
NCI_ATTR_NOCACHE, (NCIDuration)0);

nci_stmt_execute(svcctx, stmt, error, sqldroplob);

nci_stmt_execute(svcctx, stmt, error, sqlcreatelob);

/*
* 图片写入
*/
lob_offset = 0;
size = 0;

/* 以rb方式打开文件 */
if ((filepr = fopen("./cat.gif", "rb")) == NULL)
{
    printf("filepr: 文件打开发生错误!\n");
    exit(0);
}
fseek(filepr, 0, SEEK_END);
size = ftell(filepr);
fseek(filepr, 0, SEEK_SET);

int pos = 0;
int ret = 0;
while((ret = fread(bufgifw, 1, sizeof(bufgifw), filepr)) == sizeof(bufgifw))
{
    NCILobWrite(svcctx, error, lobgif, &amtp, pos, (dvoid *)bufgifw, sizeof(bufgifw), piece,\
(dvoid *)0, (sb4(*)())(dvoid *, dvoid *, ub4 *, ub1 *))0, (ub2)0, (ub1)SQLCS_IMPLICIT);
    pos += sizeof(bufgifw);
    memset(bufgifw, 0, sizeof(bufgifw));
}

NCILobWrite(svcctx, error, lobgif, &amtp, pos, (dvoid *)bufgifw, ret, piece,\
(dvoid *)0, (sb4(*)())(dvoid *, dvoid *, ub4 *, ub1 *))0, (ub2)0, (ub1)SQLCS_IMPLICIT);

fclose(filepr);

/*
* 图片读取
*/
ub4 sizeR = 0;
NCILobGetLength(svcctx, error, lobgif, &sizeR);

/* 以wb+(二进制写入)方式打开文件 */
if ((filepw = fopen("./out.gif", "wb+")) == NULL)

```

```

    {
        printf("filepw:文件打开发生错误!\n");
        exit(0);
    }

    pos = 0;

    while(sizeR - pos >= sizeof(bufferR))
    {
        NCILobRead(svcctx, error, lobgif, &amtp, pos, (void *)bufferR, sizeof(bufferR),\
            (dvoid *)0, 0, (ub2)0, (ub1)SQLCS_IMPLICIT);

        fwrite(bufferR, 1, sizeof(bufferR), filepw);
        pos += sizeof(bufferR);
        memset(bufferR, 0, sizeof(bufferR));
    }

    NCILobRead(svcctx, error, lobgif, &amtp, pos, (void *)bufferR, sizeR - pos,\
        (dvoid *)0, 0, (ub2)0, (ub1)SQLCS_IMPLICIT);

    fwrite(bufferR, 1, sizeR - pos, filepw);

    fclose(filepw);

/*
 * 清理
 */
/* 释放大对象 */
    NCILobFreeTemporary(svcctx, error, lob);
    NCILobFreeTemporary(svcctx, error, lobgif);

/* 释放lob句柄 */
    NCIDDescriptorFree(lob, NCI_DTYPE_LOB);
    NCIDDescriptorFree(lobgif, NCI_DTYPE_LOB);

    nci_logoff_disconnect(env, svcctx, stmt, error);
}

```

#### 4.1.6. 多事务

```

#include <string.h>
#include <nci.h>

CONST OraText dbname[] = "localhost:5432"; /* ip:port/dbname */
OraText username[100] = "uxdb";
CONST OraText pwd[] = "1qaz!QAZ";
int retcode = 0;

/* 调用函数NCILogon进行连接 */
sword nci_logon_connect(NCIEnv **env, NCISvcCtx **sc, NCISstmt **stmt, NCIError **error,
    CONST OraText *username,
        CONST OraText *password, CONST OraText *dbname)

```

```

{
    retcode = NCIIInitialize((ub4)NCI_DEFAULT, (dvoid *)0, (dvoid * (*)(dvoid *, size_t))0, (dvoid * (*)(dvoid *, dvoid *, size_t))0, (void (*)(dvoid *, dvoid *))0);
    if (NCI_SUCCESS != retcode)
    {
        printf("error : NCIIInitialize!\n");
        return retcode;
    }
    retcode = NCIEnvInit(env, (ub4)NCI_DEFAULT, (size_t)0, (dvoid **)0);
    if (NCI_SUCCESS != retcode)
    {
        printf("error : NCIEnvInit!\n");
        return retcode;
    }
    retcode = NCIHandleAlloc((dvoid *)env, (dvoid **)sc, (ub4)NCI_HTYPE_SVCCTX, (size_t)0, (dvoid **)0);
    if (NCI_SUCCESS != retcode)
    {
        printf("failed to allocate connection handle!\n");
        return retcode;
    }
    retcode = NCIHandleAlloc((dvoid *)env, (dvoid **)error, (ub4)NCI_HTYPE_ERROR, (size_t)0, (dvoid **)0);
    if (NCI_SUCCESS != retcode)
    {
        printf("failed to allocate error handle!\n");
        return retcode;
    }
    retcode = NCILogon(*env, *error, sc, username, strlen((char *)username), password, strlen((char *)password), dbname, strlen((char *)dbname));
    if (NCI_SUCCESS != retcode)
    {
        printf("unable to connect to the database!\n");
        return retcode;
    }
    retcode = NCIHandleAlloc((dvoid *)env, (dvoid **)stmt, (ub4)NCI_HTYPE_STMT, (size_t)0, (dvoid **)0);
    if (NCI_SUCCESS != retcode)
    {
        printf("failed to allocate statement handle!\n");
        return retcode;
    }

    return retcode;
}

/* 重新连接, 并重新分配语句句柄 */
sword nci_reconnect(NCIEnv *env, NCISvcCtx *sc, NCISstmt **stmt, NCIError *error, CONST OraText *username,
    CONST OraText *password, CONST OraText *dbname)
{

    retcode = NCIHandleFree((dvoid *)*stmt, (ub4)NCI_HTYPE_STMT);

```

```

    if (NCI_SUCCESS != retcode)
    {
        printf("failed to release statement handle!\n");
        return retcode;
    }

    retcode = NCILogoff(sc, error);
    if (NCI_SUCCESS != retcode)
    {
        printf("logout database connection failed!\n");
        return retcode;
    }
    retcode = NCILogon(env, error, &sc, username, strlen((char *)username), password, strlen((char *)pwd), dbname, strlen((char *)dbname));
    if (NCI_SUCCESS != retcode)
    {
        printf("unable to connect to the database!\n");
        return retcode;
    }

    retcode = NCIHandleAlloc((dvoid *)env, (dvoid **)stmt, (ub4)NCI_HTYPE_STMT, (size_t)0, (dvoid **)0);
    if (NCI_SUCCESS != retcode)
    {
        printf("failed to allocate statement handle!\n");
        return retcode;
    }

    return retcode;
}

/* 调用函数NCIStmtPrepare准备语句, 调用函数NCIStmtExecute执行语句 */
sword nci_stmt_execute(NCISvcCtx *sc, NCIStmt *stmt, NCLError *error, char *sql)
{
    retcode = NCIStmtPrepare(stmt, error, (text *)sql, (ub4)strlen(sql), (ub4)NCI_NTV_SYNTAX, (ub4)NCI_DEFAULT);
    if (NCI_SUCCESS != retcode)
    {
        printf("failed to prepare sql!\n");
        return retcode;
    }
    retcode = NCIStmtExecute(sc, stmt, error, (ub4)1, (ub4)0, (NCISnapshot *)NULL, (NCISnapshot *)NULL, (ub4)NCI_DEFAULT);
    if (NCI_SUCCESS != retcode)
    {
        printf("failed to execute sql!\n");
        return retcode;
    }
}

/* 调用函数NCILogoff断开数据库连接, 并释放环境、连接、语句、错误句柄 */
sword nci_logoff_disconnect(NCIEnv *env, NCISvcCtx *sc, NCIStmt *stmt, NCLError *error)
{
    retcode = NCIHandleFree((dvoid *)stmt, (ub4)NCI_HTYPE_STMT);
}

```



```
if (NCI_SUCCESS != retcode)
{
    printf("failed to release statement handle!\n");
    return retcode;
}

retcode = NCILogoff(sc, error);
if (NCI_SUCCESS != retcode)
{
    printf("logout database connection failed!\n");
    return retcode;
}

retcode = NCIHandleFree((dvoid *)sc, (ub4)NCI_HTYPE_SVCCTX);
if (NCI_SUCCESS != retcode)
{
    printf("failed to release connection handle!\n");
    return retcode;
}

retcode = NCIHandleFree((dvoid *)env, (ub4)NCI_HTYPE_ENV);
if (NCI_SUCCESS != retcode)
{
    printf("failed to release environment handle!\n");
    return retcode;
}

retcode = NCIHandleFree((dvoid *)error, (ub4)NCI_HTYPE_ERROR);
if (NCI_SUCCESS != retcode)
{
    printf("failed to release error handle!\n");
    return retcode;
}

return retcode;
}

int main(int argc, char **argv)
{
    NCIEnv *env = NULL;
    NCISvcCtx *sc = NULL;
    NCISstmt *stmt = NULL;
    NCIErr *error = NULL;
    NCITrans *tran1 = NULL;
    NCITrans *tran2 = NULL;
    int c1 = 0;

    retcode = nci_logon_connect(&env, &sc, &stmt, &error, username, pwd, dbname);

    /* 使用主事务创建表t1 */
    retcode = nci_stmt_execute(sc, stmt, error, (char *)"create table t1(c1 int);");
```

```
/* 提交主事务 */

retcode = NCITransCommit(sc, error, NCI_DEFAULT);

/* 分配事务句柄1, 并附加至连接 */
retcode = NCIHandleAlloc(env, (void **)&tran1, NCI_HTYPE_TRANS, 0, 0);

retcode = NCIAttrSet(sc, NCI_HTYPE_SVCCTX, tran1, 0, NCI_ATTR_TRANS, error);

/* 开启新事务1 */
retcode = NCITransStart(sc, error, 0, NCI_TRANS_NEW);

/* 使用事务1插入数据 */
retcode = nci_stmt_execute(sc, stmt, error, (char *)"insert into t1 values(1);");
/* 分离事务1 */
retcode = NCITransDetach(sc, error, NCI_DEFAULT);

/* 分配事务句柄2, 并附加至连接 */
retcode = NCIHandleAlloc(env, (void **)&tran2, NCI_HTYPE_TRANS, 0, 0);

retcode = NCIAttrSet(sc, NCI_HTYPE_SVCCTX, tran2, 0, NCI_ATTR_TRANS, error);

/* 开启新的可串行事务2 */
retcode = NCITransStart(sc, error, 0, NCI_TRANS_NEW | NCI_TRANS_SERIALIZABLE);

/* 使用可串行事务2查询表t1 */
retcode = nci_stmt_execute(sc, stmt, error, (char *)"select * from t1;");

/* 插入未提交查询不到数据 */
retcode = NCISmtFetch(stmt, error, 1, NCI_FETCH_NEXT, NCI_DEFAULT);

/* 分离事务2 */
retcode = NCITransDetach(sc, error, NCI_DEFAULT);

/* 附加事务1 */
retcode = NCIAttrSet(sc, NCI_HTYPE_SVCCTX, tran1, 0, NCI_ATTR_TRANS, error);
/* 提交事务1 */
retcode = NCITransCommit(sc, error, NCI_DEFAULT);

/* 附加事务2 */
retcode = NCIAttrSet(sc, NCI_HTYPE_SVCCTX, tran2, 0, NCI_ATTR_TRANS, error);
```

```
/* 使用可串行事务2查询表t1 */
retcode = nci_stmt_execute(sc, stmt, error, (char *)"select * from t1;");

/* 由于是可串行事务不会发生幻读 */
retcode = NCISmtFetch(stmt, error, 1, NCI_FETCH_NEXT, NCI_DEFAULT);

/* 清理环境 */
retcode = nci_stmt_execute(sc, stmt, error, (char *)"drop table if exists t1,t2,t3;");

retcode = NCITransCommit(sc, error, NCI_DEFAULT);

retcode = NCISmtFree(tran1, NCI_HTYPE_TRANS);

retcode = NCISmtFree(tran2, NCI_HTYPE_TRANS);

retcode = nci_logoff_disconnect(env, sc, stmt, error);

return 0;
}
```