槽的令牌上。

密码设备可以按照某一命令集执行某些密码操作，这些命令通常要经过标准设备驱动程序，例如 PCMCIA 卡服务程序或槽服务程序。Cryptoki 使每个密码设备看起来逻辑上很像其他设备，而不管是用什么技术实现的。因此，应用程序不必直接与设备驱动器接口（或甚至不必知道包括哪些设备）交互，Cryptoki 隐藏了这些细节。另外，基础设备完全能用软件来实现（例如，在一个服务器上运行的处理程序），不需要专用硬件。

Cryptoki 可以作为支持接口功能的库来实现，而应用程序则与该库连接。应用程序可以直接与 Cryptoki 连接，或者 Cryptoki 是一个“共享”库（或动态连接库），在这种情况下，应用程序动态地连接库。为了防止动态库被替换掉而带来泄密风险，从安全角度来说，一般建议直接连接该库。

设备的种类和所支持的能力的种类将取决于专用 Cryptoki 库。PKCS#11 标准只定义库的接口，不定义库的特征。要特别指出的是，并不是所有的库都支持这个接口（因为不是所有的令牌支持所有的机制）中定义的机制（算法），并且库也许只支持可使用的所有密码设备的一个子集。

#### 12.2.1.2　令牌逻辑视图

Cryptoki 的令牌逻辑视图是一个能存储对象和能执行密码函数的设备。Cryptoki 定义了 3 个对象：数据、证书和密钥。数据对象由应用程序定义。一个证书对象存储一个证书，一个密钥对象存储一个密钥。密钥可以是一个公开密钥、一个私有密钥或是一个对称密钥，每个种类的密钥在专用机制中使用其定义的辅助类型。令牌的这种逻辑视图如图 12-3 所示。

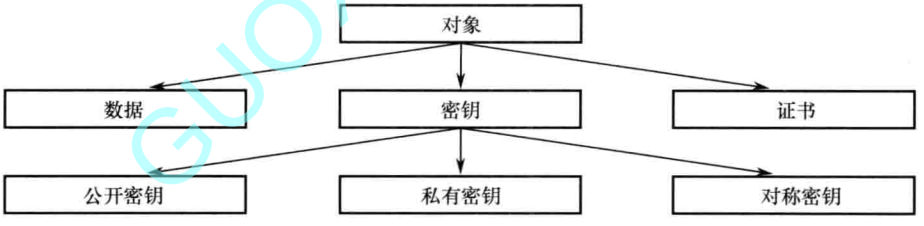


图 12-3　令牌逻辑视图

对象根据它们的使用期限和可见度进行分类。“令牌对象”能被所有与之连接并有足够权限的应用程序访问。即使关闭应用程序与令牌之间的“会话”和令牌从其槽中移除后，“令牌对象”仍保存在令牌上。而“会话对象”更具有暂时性，当以任何一种方式关闭会话时，由该会话产生的所有会话对象就会自动被销毁。此外，会话对象只能由产生它们的应用程序访问。

#### 12.2.1.3　Cryptoki 的几种常用概念

为了便于理解，下面归纳介绍了 Cryptoki 几种常用的概念。

（1）槽

槽是一个逻辑读卡器，若一台密码设备存在于读卡器中，就说一个令牌存在于该槽中。

（2）令牌

令牌是一个能存储对象、执行密码功能的设备。

（3）会话

会话是应用程序和令牌之间的逻辑连接。打开一个会话后，应用程序便访问令牌的公共对象。概括来讲，一个打开的会话能用来执行 3 类操作：管理操作（如注册）、对象管理操作（如在令牌上建立或销毁一个对象）和密码操作（如计算一个消息摘要）。

（4）对象

对象是属性的集合，对象可根据生命周期分为会话对象和令牌对象。应用程序打开一个会话时，Cryptoki 以会话句柄方式标识会话，便于应用程序使用会话句柄访问会话，同时应用程序可通过调用 Cryptoki 接口创建或查询对象，而 Cryptoki 以对象句柄方式标识对象，便于会话通过对象句柄访问对象。

（5）密钥

密钥可以是一个公钥、一个私钥或一个对称密钥。

（6）机制

机制详细说明某种加密过程是如何执行的。一个机制作为一个对象，明确指定了一个密码处理是如何执行的。Cryptoki 中定义的机制被不同的密码操作支持。对于一个特定的令牌，一个特定的操作只支持 Cryptoki 中定义的机制集合的一个子集。

（7）属性

属性描述了对象的特征。在创建对象时，Cryptoki 对哪些属性必须指定、哪些可选都有明确的规定，并且明确了属性之间的一致性定义。在对象复制时，能否修改对象的属性依赖于确定对象的属性值，如一个保密密钥对象在使用 C_CopyObject 进行对象复制时其 CKA_SENSITIVE 属性的值可以由 CK_FALSE 改变为 CK_TRUE，而不能由 CK_TRUE 改变为 CK_FALSE。

（8）用户类型

Cryptoki 识别 3 种用户类型：管理员、普通用户、上下文特定用户。只有普通用户在通过认证之后才能访问令牌上的私有对象。

管理员的角色是初始化一个令牌，并设置普通用户的 PIN（口令），或许还要操作一些公用对象。普通用户在管理员没有为其设置 PIN 之前是不能登录令牌的。事实上，管理员和普通用户可以是一个人，这可根据密码令牌的管理策略来确定。上下文特定用户只有在重新认证密码操作时才会用到。

（9）会话句柄和对象句柄

一个会话句柄是一个 Cryptoki 赋予的会话值。在许多方式中它类似于一个文件句柄。对于函数来说，它被规定用来确定函数应当执行哪一个会话。一个应用程序的所有线程能平等地访问所有的会话句柄。

对象句柄是用来控制 Cryptoki 对象的标识符。与会话句柄类似，一个对象句柄的可见性在一个应用程序的所有线程中都是相同的。只读会话仅能对令牌对象进行只读访问，读写会话对令牌对象能进行读/写访问。

#### 12.2.1.4 Cryptoki 前缀说明

为了便于标记，Cryptoki 对函数、类型定义、常量、变量都增加了标识前缀，如表 12-2 所示。

表 12-2　Cryptoki 前缀说明

| 前　缀 | 说　明 | 前　缀 | 说　明 |
|---|---|---|---|
| C_ | 函数 | CKP_ | 伪随机功能 |
| CK_ | 数据类型或通用常数 | CKS_ | 会话状态 |
| CKA_ | 属性 | CKR_ | 返回值 |
| CKC_ | 证书类型 | CKU_ | 用户种类 |
| CKF_ | 位标志 | CKZ_ | Salt/Encoding 参数资源 |
| CKG_ | 屏蔽生成功能 | H | 句柄 |
| CKH_ | 硬件特征类型 | Ul | CK_ULONG |
| CKK_ | 密钥类型 | P | 指针 |
| CKM_ | 机制类型 | Pb | 指向 CK_BYTE 的指针 |
| CKN_ | 通知 | Ph | 指向句柄的指针 |
| CKO_ | 对象级别 | Pul | 指向 CK_ULONG 的指针 |

#### 12.2.1.5　调用 Cryptoki 的一般过程

为了完成一次密码操作，需要经过：①初始化环境；②获取槽列表；③打开会话；④用户身份登录；⑤进行相关操作。完成操作后，需要：⑥登出；⑦关闭会话；⑧清空环境。

**1．初始化环境函数**

```
CK_DEFINE_FUNCTION(CK_RV, C_Initialize)(
    CK_VOID_PTR pInitArgs);
```

C_Initialize 初始化 Cryptoki 库。pInitArgs 取值为 NULL_PTR，或为指向含有说明库应该如何处理多线程访问信息的 CK_C_INITIALIZE_ARGS 结构。

**2．获取槽列表函数**

```
CK_DEFINE_FUNCTION(CK_RV, C_GetSlotList)(
    CK_BBOOL tokenPresent,
    CK_SLOT_ID_PTR pSlotList,
    CK_ULONG_PTR pulCount);
```

C_GetSlotList 用于获得系统中的一个槽列表。tokenPresent 表明所获得的列表是否只包括带有一个当前令牌（TRUE）的那些槽，或者包括所有槽（FALSE）；pulCount 指向接收槽数量的单元。

应用程序调用 C_GetSlotList 有两种方法：

① 如果 pSlotList 是 NULL_PTR，那么 C_GetSlotList 所能做的一切是返回（in*pulCount）槽的数量，而不是实际返回一个槽列表。在 C_GetSlotList 入口上由 pulCount 指向的缓冲区的内容此时无意义，该调用返回值 CKR_OK。

② 如果 pSlotList 不是 NULL_PTR，那么*pulCount 必须包含由 pSlotList 所指的缓冲区的大小（以 CK_SLOT_ID 单元为单位）。如果该缓冲区有足够大的空间容纳槽列表，那么在该缓冲区中返回该列表，并返回 CKR_OK。反之，调用 C_GetSlotList 返回值 CKR_BUFFER_TOO_SMALL。在任何一种情况下，值*pulCount 被设置为可容纳槽数的大小。

由于 C_GetSlotList 不分配自己的空间，因此应用经常两次调用 C_GetSlotList。然而，多次调用 C_GetSlotList 并不是必需的。

C_GetSlotList 报告的所有槽必须能够被 C_GetSlotInfo 有效查询。另外，通过 Cryptoki 库可访问的槽的设置在调用 C_Initialize 时就被确定。如果一个应用调用 C_Initialize 和 C_GetSlotList，那么用户接入一个新的设备，如果再次调用 C_GetSlotList 的话，该设备不能突然一下子作为一个新槽出现。为了识别新的设备，需要首先调用 C_Initialize（为了能够成功调用 C_Initialize，需要首先调用 C_Finalize）。即使 C_Initialize 被成功调用，新设备也有可能不能被成功识别。在某些平台上，可能需要重新启动整个系统。

### 3．打开会话函数

```
CK_DEFINE_FUNCTION(CK_RV, C_OpenSession)(
    CK_SLOT_ID slotID,
    CK_FLAGS flags,
    CK_VOID_PTR pApplication,
    CK_NOTIFY iNotify,
    CK_SESSION_HANDLE_PTR phSession);
```

C_OpenSession 打开某一特定槽中应用和令牌间的会话。slotID 是槽的 ID；flags 表明会话的类型；pApplication 是一个要传递给通知回调的应用定义的指针；iNotify 是通知回调函数的地址；phSession 指向接收新会话句柄的单元。

当打开一个带 C_OpenSession 的会话时，标志参数由 CK_SESSION_INFO 数据类型中定义的零位或多位标志的逻辑或组成。由于传统原因，必须总是设置 CKF_SERIAL_SESSION 位；如果调用 C_OpenSession 时该位没有设置，调用则不成功，返回错误码 CKR_PARALLEL_NOT_SUPPORTED。

一个应用程序可能并行打开的会话数量会有一个限制，它取决于会话是“只读”还是“读/写”。尝试打开一个由于有太多某种类型的现有会话而不能成功时，会返回 CKR_SESSION_COUNT。

如果令牌是写保护的（如 CK_TOKEN_INFO 结构中所说），那么可以打开带令牌的只读会话。

如果调用 C_OpenSession 的应用程序已经有带令牌的管理员打开的读/写会话，那么任何试图打开带令牌的只读会话均失败，返回错误码 CKR_SESSION_READ_WRITE_SO_EXISTS。

Cryptoki 采用 iNotify 回调函数来指明某些事件的应用。如果应用不希望支持回调，则应该传输一个 NULL_PTR 值作为 iNotify 参数。

### 4．用户身份登录函数

```
CK_DEFINE_FUNCTION(CK_RV, C_Login)(
    CK_SESSION_HANDLE hSession,
    CK_USER_TYPE userType,
    CK_CHAR_PTR pPin,
    CK_ULONG ulPinLen);
```

C_Login 为用户注册一个令牌。hSession 是会话句柄；userType 是用户类型；pPin 指向用户的 PIN；ulPinLen 是 PIN 的长度。

如果调用成功，则根据用户的类型，应用的各个会话要么进入“读/写管理员函数”状态，要么进入“只读用户函数”状态。

如果正如 CK_TOKEN_INFO 中的 CKF_PROTECTED_AUTHENTICATION_PATH 标志所表明的，令牌有一条“受保护的认证路径”，那就意味着应用不需要通过 Cryptoki 库发送 PIN，用户也有某种办法能让其被令牌认证。其中一种可能的方法是用户进入令牌自身的或槽设备的 PIN 键盘上的 PIN。或者用户可能甚至不使用 PIN——比如说，可以由某种指纹阅读设备取得认证。为了注册一个带受保护认证的路径，C_Login 的 pPin 参数应该是 NULL_PTR。当 C_Login 返回时，令牌支持的任何方法都将被执行；返回值 CKR_OK 说明用户已被成功认证，返回值 CKR_PIN_INCORRECT 则说明用户被拒绝访问。

### 5．登出函数

```
CK_DEFINE_FUNCTION(CK_RV, C_Logout)(
    CK_SESSION_HANDLE hSession);
```

C_Logout 将用户从令牌中注销。hSession 是会话的句柄。

如果调用成功，应用的每个会话根据当前的用户类型或者进入“读/写公共会话”状态，或者进入“只读公共会话”状态。

当 C_Logout 成功执行后，专用对象的任何应用的句柄就都无效了（即使用户以后又返回注册到令牌，那些应用仍然是无效的）。另外，属于应用的会话的所有专用会话对象都被破坏。

如果应用程序的会话中有任何活动密码操作或对象查找的操作，而且接着该应用成功执行 C_Logout，那么那些操作也有可能不再是有效的。因此，注销前应当结束任何正在执行的操作。

### 6．关闭会话

```
CK_DEFINE_FUNCTION(CK_RV, C_CloseSession)(
    CK_SESSION_HANDLE hSession);
```

C_CloseSession 关闭应用和令牌间的会话。hSession 是会话的句柄。

当关闭一个会话时，会话创建的所有会话对象均被自动销毁，即使应用还有其他会话在“使用”该对象。

如果该函数执行成功，关闭了应用程序和令牌间的最后一个会话，那么应用令牌的逻辑状态返回到公共会话。应用程序打开令牌的任何新的会话或者是只读公共会话或者是读/写公共会话。

尽管假设该 C_CloseSession 关闭一个会话，返回值 CKR_SESSION_CLOSED 是一个错误的返回。它事实上表明（有时不可能）在该函数正在执行时，应用程序又调用 C_CloseSession 来关闭该会话，并且该调用首先结束执行。

### 7．清空环境函数

```
CK_DEFINE_FUNCTION(CK_RV, C_Finalize)(
    CK_VOID_PTR pReserved);
```

调用 C_Finalize 表明，用 Cryptoki 库完成了一个应用程序。它应当是应用程序所做的最后一次调用。pReserved 参数为未来版本使用。对于该版本，它应当被设置成 NULL_PTR。