

开发 Engine 应用程序的基本原理

2

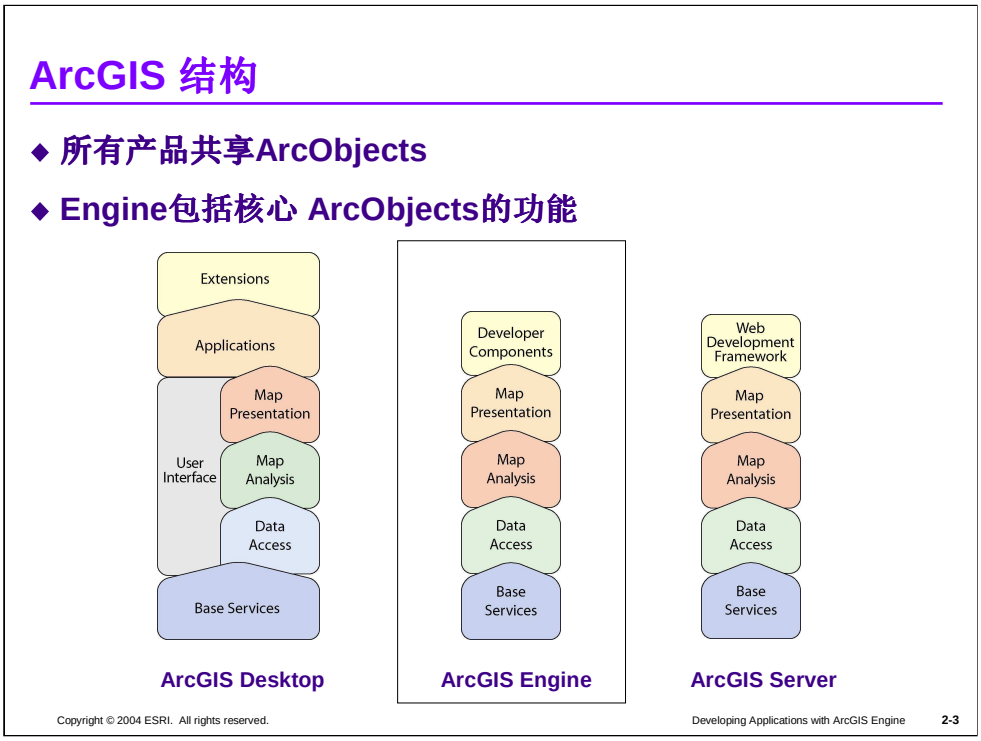
Copyright © 2004 ESRI. All rights reserved.

Developing Applications with ArcGIS Engine

课程2

- ArcObjects libraries
- ArcObjects Engine APIs
- 开发 Engine 应用程序
- 许可

Copyright © 2004 ESRI. All rights reserved.



ArcGIS architecture

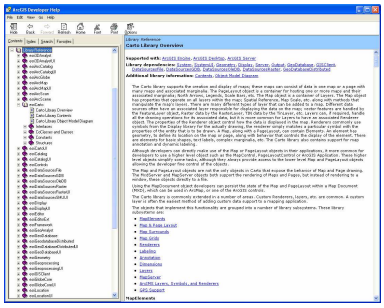
The ArcGIS system is based on a common library of shared GIS components called *ArcObjects*. ArcObjects include a wide variety of programmable components ranging from fine-grained objects, such as individual geometry objects, to coarse-grained objects, such as a map object, that can used to interact with existing ArcMap documents. You, as a developer, can work with ArcObjects using standard programming frameworks to extend ArcGIS Desktop, build custom applications with ArcGIS Engine, and implement enterprise GIS applications using ArcGIS Server. The product diagrams show that the objects within broad categories (e.g., Base Services, Data Access, Map Analysis, and Map Presentation) are contained in all three products. These four categories contain the majority of the GIS functionality exposed to developers and users in ArcGIS.

This commonality of function between all the products is important for developers to understand, since it means that when working in a particular category, much of the development effort can be transferred between the ArcGIS products with little change to the software. After all, this is exactly how the ArcGIS architecture is developed. Code reuse is a major benefit of building a modular architecture.

ArcGIS Engine is a complete set of embeddable GIS components for developers to build custom applications. This class focuses on building and deploying custom applications using ArcGIS Engine.

ArcGIS 库

- 所有产品共享库
- 一些特定库不被共享
  - 例如: esriArcMapUI
- 明确哪些库是属于不同产品
- 理解库的互依赖性
- 开发工具
  - ArcGIS Developer Help
  - ESRI 自动提及的附加资源



Copyright © 2004 ESRI. All rights reserved.

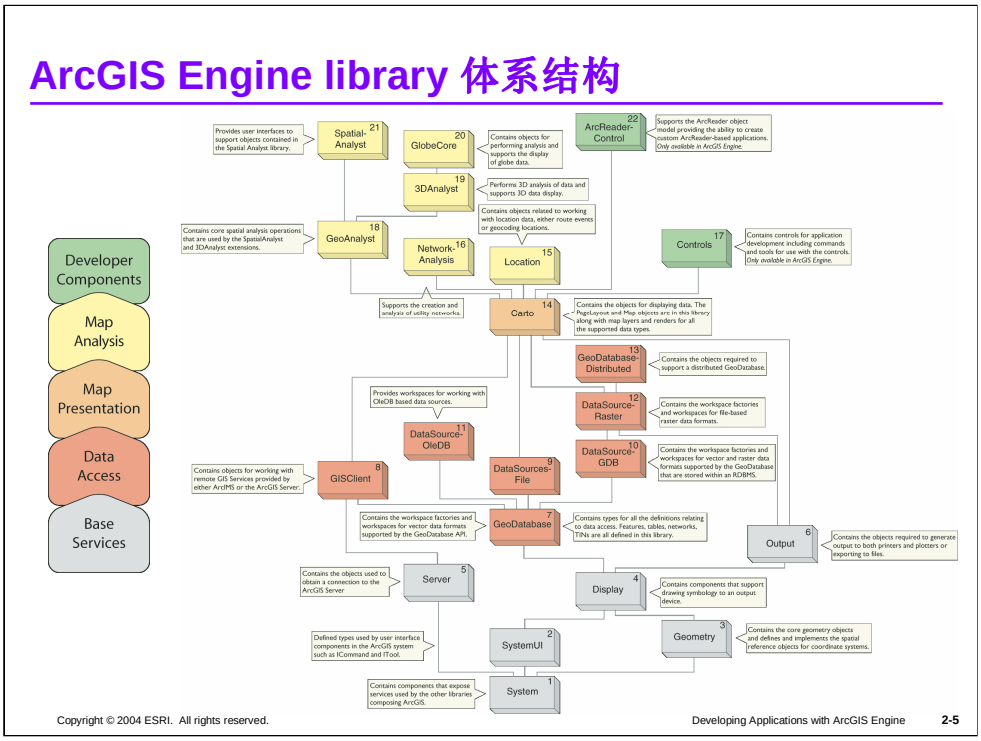
ArcGIS libraries

The ArcGIS 9 architecture is divided into a number of libraries. It is possible for a library to have any number of DLLs and EXEs within it, and it stores type information for ArcObjects components. The requirements that components must meet to be within a library are well-defined. For instance, a library such as `esriGeometry` (from the Base Services set of modules) has the requirement of being thread safe, scalable, with no user interface components, and deployable on a number of computing platforms.

Some of these libraries are shared among ArcGIS Desktop, Engine, and Server. For example, all three products share libraries such as: `System`, `SystemUI`, `Geometry`, `Geodatabase`, and `Carto` to name a few. The libraries ending in UI are designated ArcGIS Desktop libraries. The UI represents the "user interface" interaction you get with Desktop. Even though the `systemUI` library ends with UI, it is still provided in all the products.

It is important to understand that the ArcObjects libraries have dependencies upon one another. For example, if you searched for "Display Library Overview" in the Developer Help, you would see a listing of library dependencies for the `Display` library. This library is dependent upon the `System`, `SystemUI`, and `Geometry` libraries in that order. You will notice that all libraries are dependent upon the `system` library. This library is the lowest level in the ArcGIS architecture. The library contains components that expose services used by the other libraries composing ArcGIS.

ESRI provides you an "Automatic Reference" add-in when working in VB 6 and .NET. This add-in provides a quick and easy way of referencing the libraries needed for the specific application you are creating. You can also determine which library your interface or class belongs to by using the Library Locator tool or by using the Developer Help. The products are broken down with their respective libraries. Each library has a listing of all interfaces and classes.

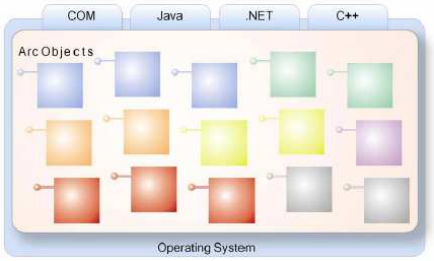


Library architecture

You may be used to working with ArcObjects libraries (such as esriFramework) that are not listed here. Those libraries are not part of Engine. The online Developer Help system lists which libraries are and are not part of ArcGIS Engine.

ArcObjects

- 所有ArcGIS产品都是由ArcObjects组成
- ArcObjects都是C++对象
- 通过不同的应用程序接口（API）来进行调用
- 本身其就是COM组成



Copyright © 2004 ESRI. All rights reserved.

ArcObjects

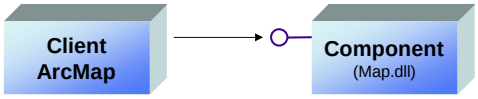
As stated earlier, the ArcGIS system is based on ArcObjects components. ArcObjects is a set of platform-independent software components, written in C++, that provide services to support GIS applications on the desktop, in the form of thick and thin clients, and on the server.

In addition to C++, ArcObjects makes use of the Microsoft Component Object Model (COM). COM is often thought of as simply specifying how objects are implemented and built in memory and how these objects communicate with one another. While this is true, COM also provides a solid infrastructure at the operating system level to support any system built using COM. On Microsoft Windows operating systems, the COM infrastructure is built directly into the operating system. For operating systems other than Microsoft Windows, this infrastructure must be provided for the ArcObjects system to function.

The illustration above shows different APIs such as COM, .NET, C++, and JAVA. If you are developing in Desktop, you cannot work in the Java API. This is meant solely for developers in ArcGIS Engine or Server. You can see though that regardless of what API you use, the underlying premise remains the same: ArcObjects is based off of COM and the language will need to be COM-compliant. For example, if you develop in .NET, you will still need to create COM components, so there will need to be some sort of interoperability involved.

COM

- 连接一个软件模型和另一个软件模型的一种协议
  - 独立的一种语言
  - 二进制规范
  - 运用界面或组件来开发、配置程序模型
- 客户端调用服务器并且服务给予响应
- 所有通讯是通过接口进行的



Copyright © 2004 ESRI. All rights reserved.

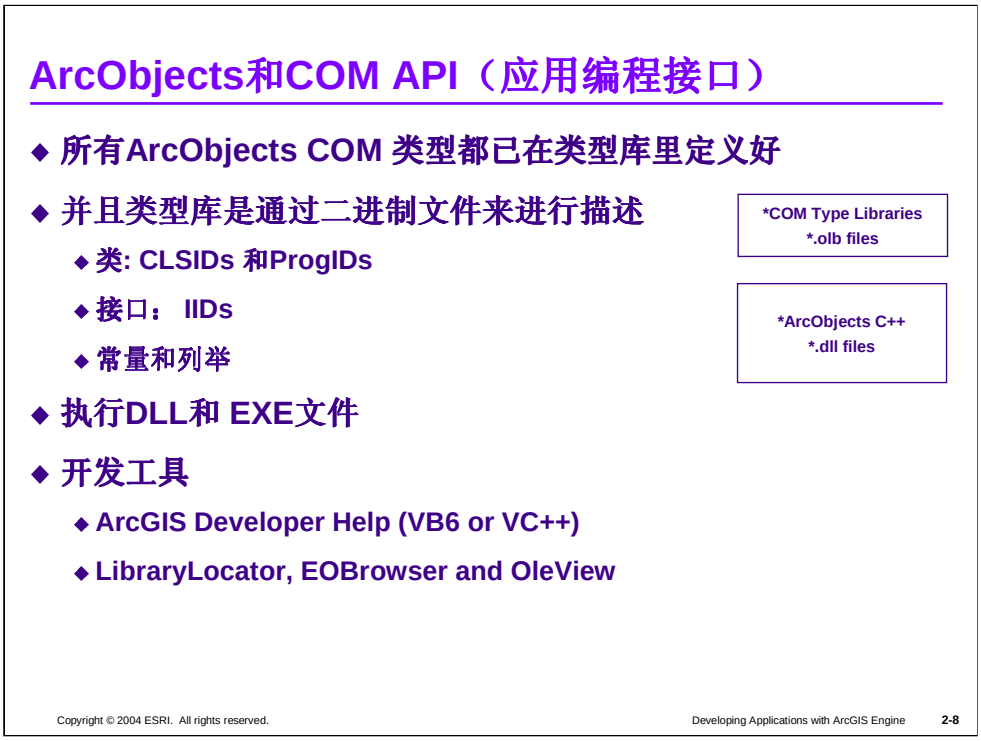
COM

COM components can be clients and servers. The server provides functionality, and the client uses that functionality. COM facilitates the communication between these two types of components. An individual component can act as both a client and a server.

In a COM system, the client, or user of functionality, is completely isolated from the server, or provider of that functionality. The client only needs to know that the functionality is available. With that knowledge, the client can make calls to the server and expect the server to honor them. In this way, COM acts as a contract between client and server. If the server breaks that contract, the behavior of the system will be unspecified. Therefore, COM development is based on trust between the functionality provider and the user.

ArcObjects contains many servers that provide, via their interfaces, thousands of properties and methods. A developer who uses ArcObjects can assume that all these properties and interfaces have been fully implemented.

In the example above, a client, such as ArcMap, may call on Map.dll via an interface for a particular service to render layers in the map.



Type libraries

A type library is a binary description of all coclasses, interfaces, methods, and types contained within a server or servers. By utilizing standard COM interfaces, various development tools and compilers can gain information about the coclasses and interfaces supported by a particular library. In order to support the concept of a language-independent development set of components, all relevant data concerning the ArcObjects libraries are shipped inside type libraries. There are no header files, source files, or object files supplied or needed by external developers.

All ArcGIS type libraries are located in the \Program Files\ArcGIS\com folder, and actual implementations of ArcObjects components are found in DLL files that are in the \Program Files\ArcGIS\bin folder.

Developer tools

There are several developer tools that allow you to get additional information on coclasses and interfaces when developing with ArcObjects components. The object model diagrams are provided in the PDF files included with the software distribution. The PDF files are searchable; you can use the Search dialog box in Acrobat Reader to find classes and interfaces quickly.

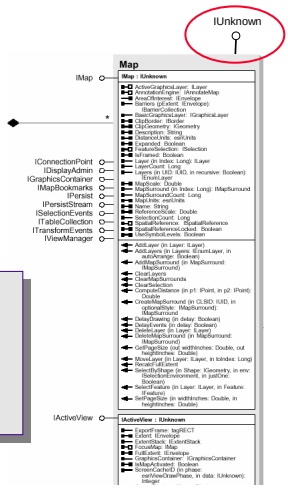
In addition to PDF files, the type library information can be viewed in a number of object browsers, depending on your development platform. Visual basic and .NET have built-in object browsers; OleView also displays type library information. The best object viewer can be used to view type information is the ESRI object viewer. This object viewer can be used to view type information for any type library that you reference within it.

All interfaces and coclasses are also documented in the component help file. Ultimately, this will be the help most commonly accessed when you get to know the object models better. For Visual Basic and .NET developers, this is a compiled HTML file that can be viewed by itself or when using an IDE.

COM 类

- COM类执行接口
- 所有类执行 IUnknown
  - 管理所有对象的产生、终结
  - 成员: QueryInterface, AddRef 和 Release
- Coclasses 是可创建的

```
Public Sub mapName()
    Dim pMap As IMap
    '你能通过一个接口来创建一个实例
    Set pMap = New Map
End Sub
```



Copyright © 2004 ESRI. All rights reserved.

COM classes

Developing with COM means developing using interfaces implemented on COM components. All communication between objects is made via their interfaces. COM interfaces are abstract, meaning there is no implementation associated with interfaces; codes associated with interfaces come from a class implementation.

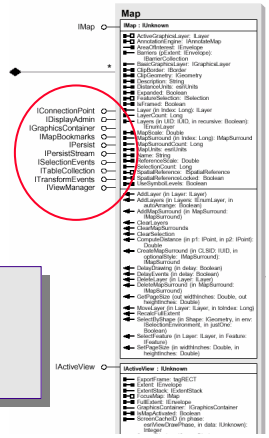
COM classes provide the code associated with one or more interfaces, encapsulating the functionality entirely within the class. Two classes may have the same interface but implement the interfaces differently. By implementing these interfaces in this way, COM displays classic object-oriented polymorphic behavior. COM does not support the concept of multiple inheritance, but this is not a shortcoming, because individual classes can implement multiple interfaces.

All COM objects must implement IUnknown interface. The IUnknown interface performs two tasks: it controls object lifetime and provides runtime type support. Object lifetime is controlled with two methods: *AddRef* and *Release*. QUERYINTERFACE

To distinguish one class from another, it is often necessary to use the TypeOf operator in Visual Basic. TypeOf simply calls QueryInterface (QI) on the object and asks it if it supports the interface on the right side of the Is operator. If QI was successful, a True value is returned. In the example above, because the PageLayout and Map coclasses both support IActiveView, the TypeOf operator must be used to determine which view the current document is displaying in ArcMap.

COM 接口

- 定义一组逻辑的方法和属性
- 从 IUnknown (default)继承
- ArcObjects 支持
  - 客户化接口
  - 通过 QueryInterface (QI)访问接口




```
Public Sub mapName()
    Dim pMap As IMap
    Set pMap = New Map
    '通过定义的pMap你可以有权使用其的properties and methods
    MsgBox pMap.Name
End Sub
```

Copyright © 2004 ESRI. All rights reserved.

COM interfaces

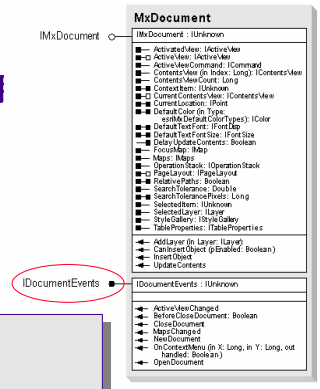
COM objects communicate with each other through COM interfaces. When working with COM objects, the developer never works with the COM object directly but gains access to the object via one of its interfaces. COM interfaces are designed to be a grouping of logically related methods and properties. The virtual functions are used by the client and implemented by the server.

With ArcObjects, most objects have a number of interfaces that a developer will work with. No object has only one interface. Because you may need functionalities that exist on different interfaces, you may need to do QueryInterface on the same object. The term *QueryInterface* refers to the process of using additional interfaces on the same object.

In the example above, the Map coclass has a number of interfaces that can be accessed. As illustrated above, only the main interface on the class is expanded on the OMD. All of the other interfaces are present, but the members are not visible to avoid cluttering the diagram. To discover the members of these other interfaces, visit the ArcObjects Developer Help.

COM 外部连接的一些接口

- 提供跟客户端来进行交换的一些事件的途径
- 通过连接对象并接收它外部接口
- 许多不同的事件在ArcObjects都是可用
  - 例如: document load events, map drawing events, editing events, …




```
Private WithEvents MxDocEvents As MxDocument

Private Sub setNewValues()
    . . .
    Set MxDocEvents = pMxDoc.FocusMap 'Now can access events
End Sub

Private Function MxDocument_NewDocument() As Boolean
    MsgBox "New ArcMap document"
End Function
```

Copyright © 2004 ESRI. All rights reserved.

COM outbound interfaces

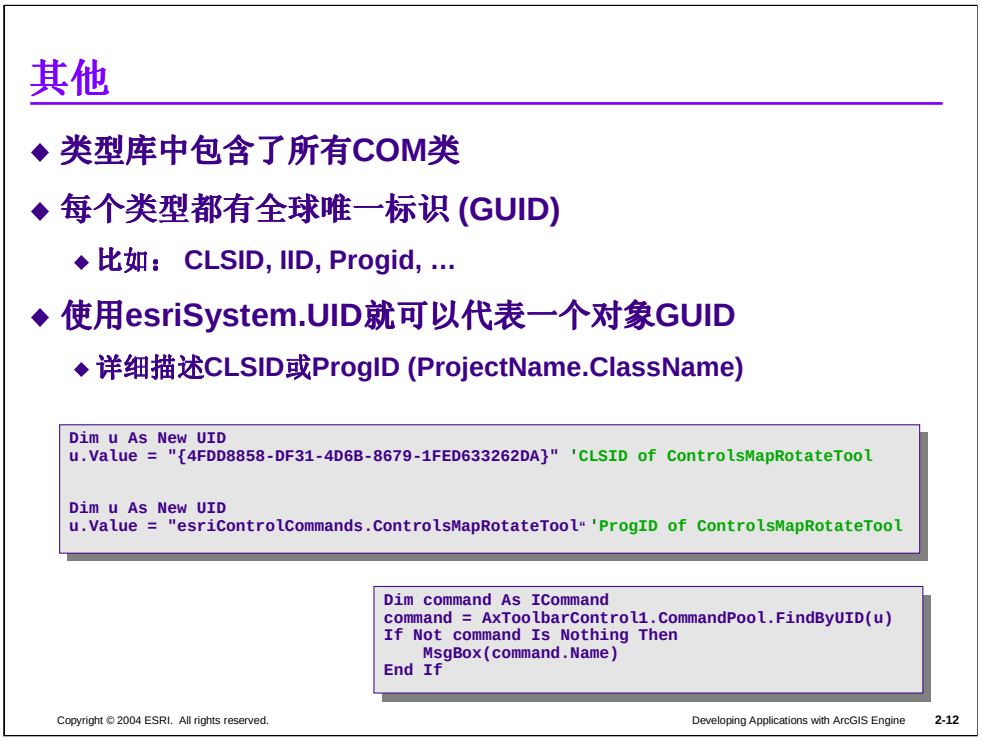
Events let you know when something has occurred. You can add code to respond to an event associated with ArcObjects components. For example, a command button has a Click event which gets triggered when the user clicks the control. Outbound interface contains events associated with a COM component.

The source COM object will fire events to an outbound interface when certain actions occur. For example the Map coclass is a source of IActiveViewEvents and will fire an ItemAdded event when a new layer is added to the map. The source object can have any number of clients (or event sink objects) listening to events. Also a source object may have more than one outbound interface; for example, the Map coclass also fires events on an IMapEvents interface.

A COM object that listens to events is said to be a sink for events. The sink object implements the outbound interface. An event sink typically uses the connection point mechanism to register its interest in the events of a source object. The connection point interfaces are the standard COM interfaces: IConnectionPoint and IConnectionPointContainer.

Through the IConnectionPoint interface, the connection point allows callers to connect a sink to the source object, to disconnect a sink, or to enumerate the existing connections. When a source object needs to inform all the sinks of a particular action, the source is said to fire an event. This results in the source iterating all the sinks and making the same method call on each. For example, when a layer is added to a map, the Map coclass fires the ItemAdded event. So all the objects listening to the Map's outbound IActiveViewEvents interface will be called on their implementation of the ItemAdded method.

IConnectionPointContainer implemented on an object makes the object "connectable" and expresses the existence of outgoing interfaces on the object. Through this interface a client may either locate a specific "connection point" for one class or it can enumerate the connection points that exist.



GUIDs and UIDs

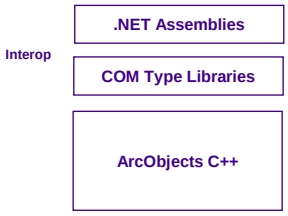
COM interfaces and coclasses are identified by a globally unique identifier (GUID). The GUID for an interface is called an interface ID (IID). The GUID for a coclass is called class ID (CLSID). A ProgID is a text alias for a CLSID. The ProgID is a string composed of the project name and the class name of the coclass.

The UID coclass can be used to represent the GUID of an object.

In the sample code above, the built in tool “ControlsMapRotateTool” can be assigned to the UID value either by using its CLSID, 4FDD8858-DF31-4D6B-8679-1FED633262DA, or its ProgID, esriControlCommands.ControlsMapRotateTool. This UID can be used later to retrieve the tool in a toolbar control.

ArcObjects和.NET API（应用编程接口）

- 所有ArcObjects类型 在 .NET 集合中都有定义
- 主要的存储方式都是 importing the COM type libraries
- PIA exposes all classes, interfaces and constants as managed .NET types
- 需要 .NET Framework 1.1
- 开发工具
  - ArcGIS Developer Help (.NET)
  - ILDASM.exe



Copyright © 2004 ESRI. All rights reserved.

.NET interoperating with COM

Code running under the .NET Framework's control is called *managed code*; conversely, code executing outside the .NET Framework is termed *unmanaged code*. COM is one example of unmanaged code. The framework promotes the interaction with unmanaged code and for COM, the technology that bridges the gap between managed and unmanaged code is called *COM Interop* and it is part of the .NET framework.

Primary Interop Assemblies (PIA)

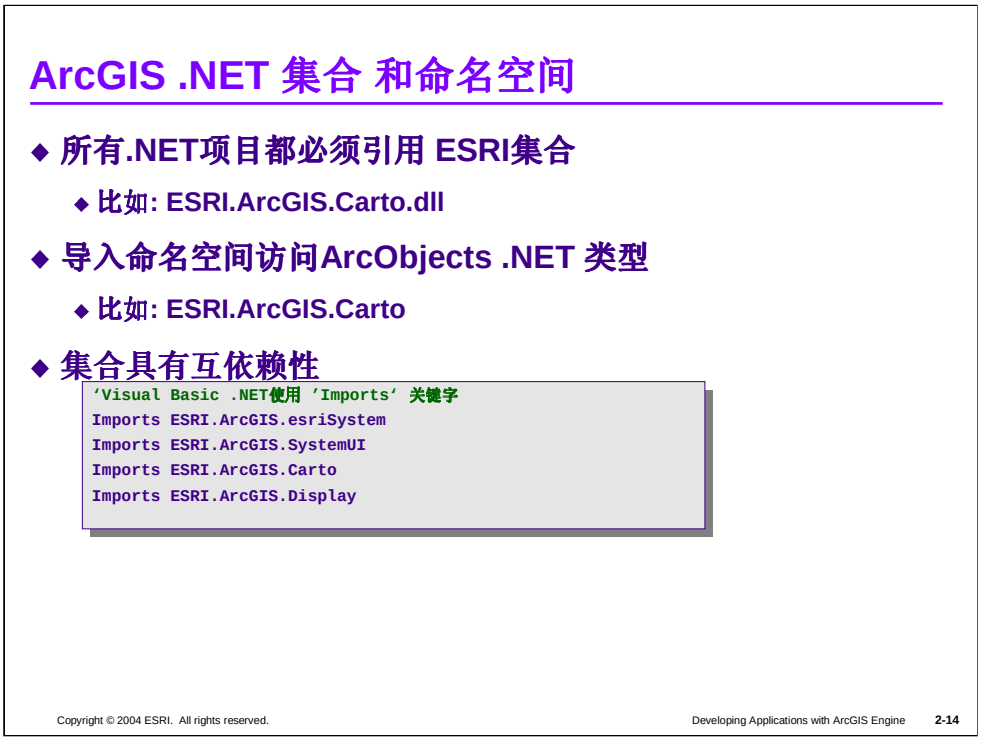
For COM Interop to work, the .NET common language runtime requires metadata for all COM types. This means COM type libraries need to be converted into .NET metadata.

PIAs are the official, vendor-supplied .NET type definitions for interoperating with underlying COM types. Primary interop assemblies are strongly named by the COM library publisher to guarantee uniqueness.

ESRI provides PIAs for all the ArcObjects type libraries that are implemented with COM. ArcGIS .NET developers should use only these interop assemblies for interoperating with ArcObjects in .NET. The ESRI primary interop assemblies are installed in the Global Assembly Cache (GAC) during install if version 1.1 of the .NET Framework is detected. ESRI supports only the interop assemblies that ship with ArcGIS. The installation directory for all assemblies is: \Program Files\ArcGIS\DotNet.

Developer tools

The Ildasm.exe is a .NET utility that shows the information inside of an assembly in a human-readable format. It shows more than just the Microsoft intermediate language (MSIL) code; it also displays the public key token, namespaces, and all types including interfaces. This utility is similar in nature to the OleView.exe utility that is used to inspect COM type libraries.



ArcGIS .NET assemblies and namespaces

Namespaces are a concept in .NET that allow objects to be organized hierarchically into logical categories of related functionality, regardless of the assembly they are defined in.

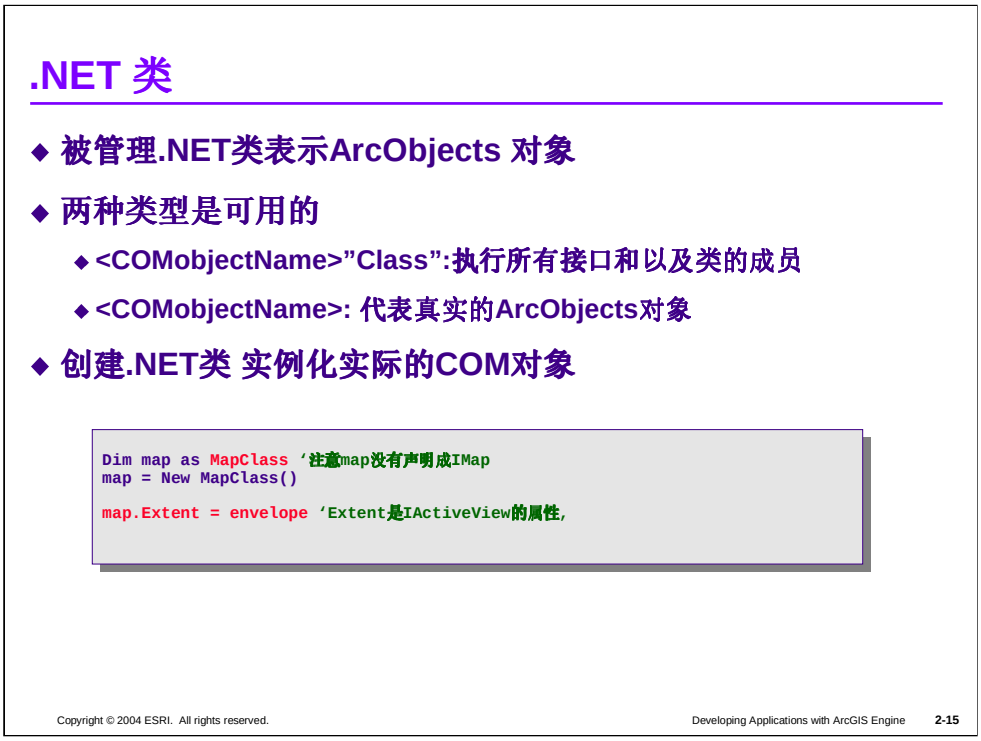
The interop assemblies are named after the object library name minus the 'esri' prefix, using the convention, for example the object library esriCarto.olb corresponds to the assembly ESRI.ArcGIS.Carto.dll.

The classes in the interop assemblies (provided as part of the ArcObjects Developer Kit for .NET) generally replicate the assembly name. For example, all the items defined in the ESRI.ArcGIS.Carto.dll assembly belong to the namespace ESRI.ArcGIS.Carto. One notable exception is the ESRI.ArcGIS.System assembly, which corresponds to the esriSystem.olb object library. To avoid a conflict with the Microsoft System root namespace, the items in this assembly belong to the ESRI.ArcGIS.esriSystem namespace.

To find out more about the contents of the ArcGIS interop assemblies, open the Contents page of the ArcGIS Developer Help system, and go to the Library Reference section.

After you have added a reference to an assembly, you can make your code simpler and more readable by adding directives that act as shortcuts when you declare or reference items specified in the namespaces contained in the referenced assemblies. This allows you to use classes and other types defined within the imported namespaces without having to fully-qualify the class and type names.

The Using and Imports statements are syntactic shortcuts not references to an assembly. These statements allow you to reference types in a namespace without a qualification. If the type is not in the mscorlib assembly that is automatically referenced by Visual Studio, you must use Add > Reference to reference the assembly.

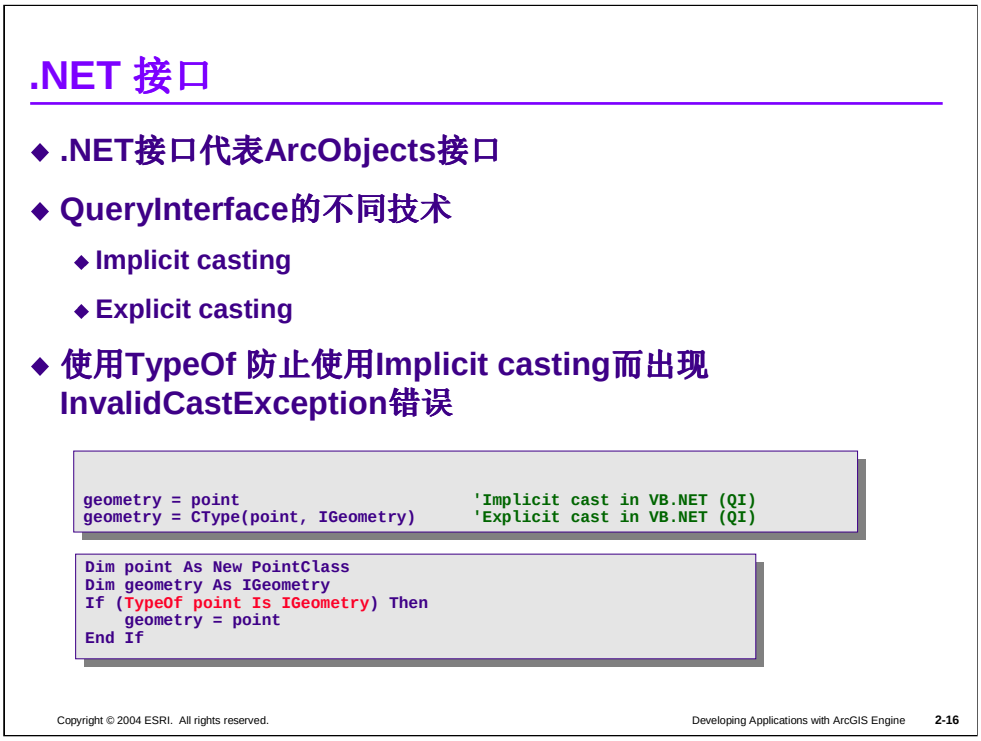


.NET classes

All COM coclasses are converted to managed classes; the managed classes have the same name as the original with ‘Class’ appended. For example, the Point coclass is PointClass. Each member of each interface the class implements is added as a class member. Any property or method a class implants can be accessed directly from the class rather than having to cast to a specific interface. This means that you are not required to QI to other interfaces as in other languages, such as VB6.0.

Although you can use the xxxClass types in almost every case, you may want to continue to use the managed interface types so that your code is more legible.

All classes additionally have an interface with the same name as the coclass that corresponds to the default interface for the coclass. For example, the PointClass has a Point interface. The type library importer adds this interface so clients can register as event sinks.



.NET interfaces

.NET uses casting to jump from one interface to another interface on the same class. In COM, this is called *QueryInterface*. VB.NET and C# cast differently.

In VB.NET, there are two types of casts: implicit and explicit. Implicit casts do not require additional syntax, whereas explicit casts require cast operators.

When casting between interfaces, it is perfectly acceptable to use implicit casts because there is no chance of data loss as there is when casting between numeric types. However, when casts fail, an exception (System.InvalidCastException) is thrown. To avoid handling unnecessary exceptions, it is best to test if the object implements both interfaces beforehand. The recommended technique is to use the TypeOf keyword which is a comparison clause that tests whether an object is derived from or implements a particular type, such as an interface.

In C#, the best method for casting between interfaces is to use the `as` operator. Using the `as` operator is a better coding strategy than a straight cast because it yields a null on a conversion failure rather than raising an exception.

A straight cast is acceptable practice if you are absolutely certain the object in question implements both interface. If the object does not implement the interface you are attempting to get a handle to, .NET will throw an exception. A safer model to use is the `as` operator, which returns a null if the object cannot return a reference to the desired interface.

```
IGeometry geometry = point;                 // straight cast
IGeometry geometry = point as IGeometry;    // as operator
```

.NET 外接口

- 可以连接COM对象并且接收其事件使用.NET 类型
- 挂起“Event” 到外接口
  - 比如: IActiveViewEvents_Event
- 技术
  - WithEvents: 用于默认外接口
  - Delegates：用于连接非默认外接口

```
Private SelectionChanged As IActiveViewEvents_SelectionChangedEventHandler

Public Overrides Sub OnClick()
 创建一个Delegates的实例,并添加其到SelectionChanged事件
  SelectionChanged = New IActiveViewEvents_SelectionChangedEventHandler(AddressOf
  OnSelectionChanged)
  AddHandler map.SelectionChanged, SelectionChanged
End Sub

Private Sub OnSelectionChanged()                    'Event handler
    MessageBox.Show("Selection Changed")
End Sub
```

Copyright © 2004 ESRI. All rights reserved.

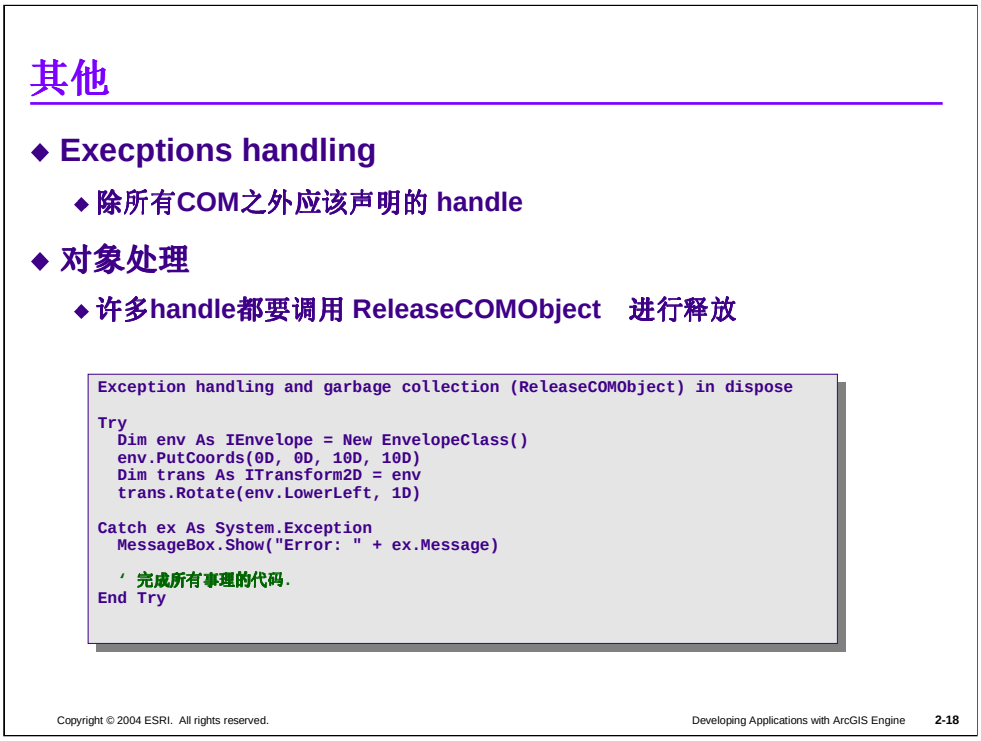
.NET outbound interfaces

The type library importer creates several types that enable managed applications to sink to events fired by COM classes. The first type is a delegate that is named after the event interface plus an underscore followed by the event name, and then the word EventHandler. For example, the SelectionChanged event defined on IActiveViewEvents interface has the following delegate defined: IActiveViewEvents_SelectionChangedEventHandler. The importer additionally creates an event interface with a “_Event” suffix added to the end of the original interface name. For example, IActiveViewEvents generates IActiveViewEvents_Event. Use the event interfaces to set up event sinks.

In event communication, the event sender class does not know which object or method will receive the events it raises. What is needed is an intermediary between the source and the receiver. The .NET Framework defines a special type (<Delegate>) that provides the functionality of a function pointer.

A delegate is a class that can hold a reference to a method. Unlike other classes, a delegate class has a signature, and it can hold references only to methods that match its signature.

To consume an event in an application, you must provide an event handler (an event-handling method) that executes program logic in response to the event and register the event handler with the event source and it must have the same signature as the event delegate.



Garbage collection

Garbage collection is .NET's method of memory management. When .NET determines that the managed heap for a process is filling up, it calls the garbage collector. The garbage collector checks variables currently in scope and removes any objects no longer referenced in the code. This process has several advantages, such as automatic memory management; however, it is important to remember that it is not deterministic. This means you know resources will be cleaned up at some point; you cannot be sure of the exact time. The garbage collector does not release resources such as Windows handles and database connections. These resources will need to be released with a destructor.

COM and garbage collection

The CLR handles reference counting for COM objects and will release the COM object sometime after it has been marked for collection. This can be problematic if a COM object needs to be released at a specific point during the program's execution. Use the Marshal.ReleaseComObject static method in the System.Runtime.InteropServices namespace to control exactly when the COM object is released.

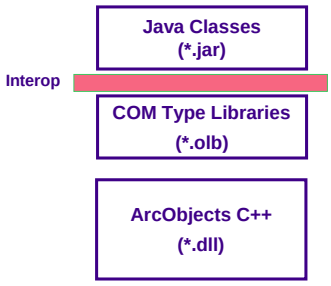
Destructors

Destructors are methods that are called when an object is destroyed. C# destructors have the same name as the class preceded by a tilde (~). The destructor method in VB .NET is called *Finalize*. Destructors are automatically called by the garbage collector, and a class should never call its destructor. In some cases, it is necessary to release critical resources rather than wait for the garbage collector. Most .NET objects that hold resources that need to be explicitly closed offer a close or dispose method that forces resource cleanup. Dispose should be called manually to release the resources, but if the client forgets to do this, garbage collection will clean up the resources.

In Visual Basic 6.0, Class_Terminate was called when the last reference to an object was dropped. This can lead to problems with circular references, which can prevent some objects from terminating correctly. In Visual Studio .NET, this problem is solved using the garbage collector. The disadvantage of using the destructor and garbage collection is that you cannot be sure when the destructor will be called.

ArcObjects和Java API

- 在ArcObjects开发包里解压缩所有Java类和接口
- 通过使用标准JAVA类型产生一个interop层访问所有COM类型
- 需要 JDK 1.4.2
- 开发工具
  - ArcGIS Developer Help (Java Doc)



Copyright © 2004 ESRI. All rights reserved.

Java API

The ArcGIS 9.0 Java API is a programming interface which interoperates with ArcObjects and is specifically designed to target Java developers. The API is not limited to any specific Java Virtual Machine or ArcGIS platform. The API exposes the functionality of ArcObjects via Java classes and interfaces, which allows Java developers to benefit from ArcObjects component reuse.

In order to develop with ArcObjects using the Java API, you must have the Java 2 Platform Standard Software Developer Kit installed. The Java API provides “Proxy” classes that are generated from ArcObjects components type libraries which allow interoperability with all of the underlying components. These proxy classes expose ArcObjects properties, methods, and events via their Java equivalents.

The ArcGIS SDK for Java is used in two ESRI developer platforms: ArcGIS Engine and ArcGIS Server.

ArcObjects Java开发包和JAR 文件

- 应用程序必须引用ArcObjects JARs
  - arcobjects.jar: Non-UI JNI equivalents
  - jintegra.jar: Runtime library that handles COM interop
  - arcgis_visualbeans.jar: UI Java beans for controls (optional)
- 导入一个或多个开发包或独立类型
  - com.esri.arcgis.<library>
  - com.esri.arcgis.carto.Map;
  - com.esri.arcgis.carto.*;

```
import com.esri.arcgis.system.*;
import com.esri.arcgis.systemUI.*;
import com.esri.arcgis.carto.*;
import com.esri.arcgis.display.*;
```

Copyright © 2004 ESRI. All rights reserved.

Java packages

The ArcGIS Engine developer kit uses standard Java Native Interface (JNI) to access core ArcObjects components. This requires some native libraries to be in the developer's path when compiling and running applications. You must be sure to include the correct paths to invoke interoperability into native ArcObjects.

All Java applications built with any of the ArcGIS developer kits must have the following JARs referenced in the respective application's classpath:

Arcobjects.jar: Contains all of the non-UI classes in one complete archive

jintegra.jar: contains all classes for the runtime library that handles interop to COM.

In addition to these, individual arcgis_xxx.jar files should be added to your classpath as needed. For example, applications that leverage the Java visual beans included with ArcObjects require that the arcgis_visualbeans.jar file be added in the classpath.

Java import statements allow fully qualified class names to be shortened to their simple names.

Java类

- Java类允许实例化并访问ArcObjects COM对象
- Classes: 不可创建的

```
/* 下列方法不能创建类/
FeatureClass fc = new FeatureClass();

/* 比如通过FeatureWorkspace创建要素类*/
IFeatureClass fc = fw.openFeatureClass("featureclass name");
```

- CoClasses: 使用“new”创建Publicly类

```
/* 直接通过NEW创建类*/
Envelope env = new Envelope();
```

Copyright © 2004 ESRI. All rights reserved.

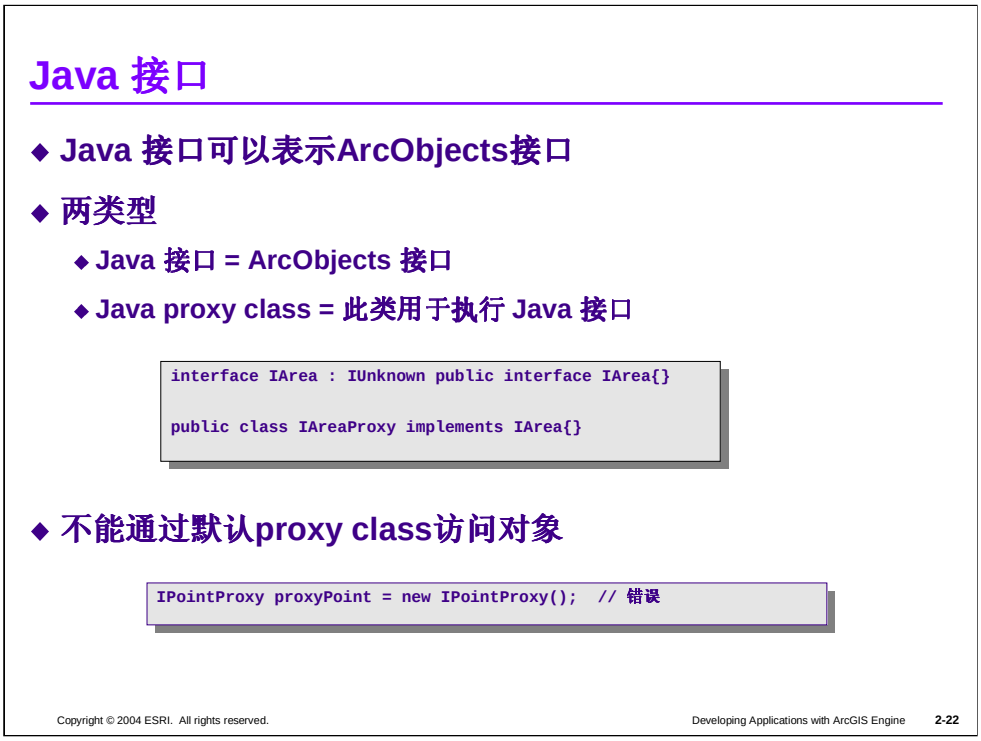
Java classes

In the ArcObjects model, classes provide the implementation of the defined interfaces. ArcObjects provides three types of classes: abstract classes, classes, and coclasses. These class types can be distinguished through the object model diagrams provided in ArcGIS Developer Help. It is important to be familiar with them before you begin to use the three class types.

In ArcObjects, an abstract class cannot be used to create new objects and are absent in the Java API for ArcObjects. These classes are specifications in ArcObjects for instances of subclasses through type inheritance. An abstract class enumerates what interfaces are to be implemented by the implementing subclass but does not provide an implementation to those interfaces. For each abstract class in ArcObjects there are subclasses that provide the implementation.

A class cannot be publicly created in ArcObjects; however, objects of this class type can be created as a property of another class or instantiated by objects from another class. In the Java API, the default constructor normally used to create a class is undefined for ArcObjects classes.

In ArcObjects, a coclass is a publicly creatable class. This means that you can create your own objects merely by declaring a new object.



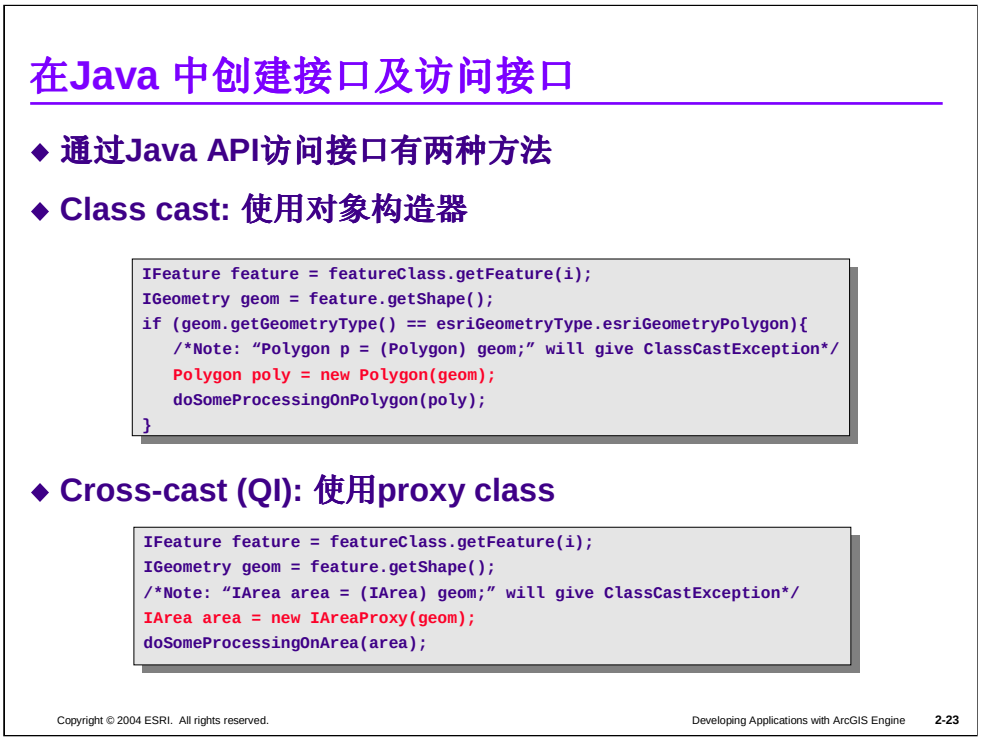
Java interfaces

ArcObjects interfaces are abstract, meaning there is no implementation associated with an interface. Objects use type inheritance; the code associated with an interface comes from the class implementation.

This model shares some features of the Java interface model. An interface in the Java language is a specification of methods that an object declares it implements. A Java interface does not include instance variables or implementation code.

The Java API has two objects for every ArcObjects interface: a corresponding interface and an interface proxy class. The interface is named in the ArcObjects style, prefixed with an I. The interface proxy class appends the term proxy to the name.

The proxy classes are used internally by the Java API to provide implementation to respective interfaces. An application developer should never use the default constructor of these classes as it holds no implementation. ArcObjects requires developers to go through an interface to access objects. The Java language does not use this model; subsequently, the Java API to ArcObjects has two ways of accessing objects: by interface or by class. You cannot access objects through the default interface proxy class.



Casting

ArcObjects follows an interface-based programming style. Many methods use interface types as parameters and have interfaces as return values. When the return value of a method is an interface type, the method returns an object implementing that interface. When a method takes an interface type as parameter, it can take in any object implementing that interface. This style of programming has the advantage that the same method can work with many different object types, provided they all implement the said interface.

Casting is used to convert between types. There are three types of potential casts you, as a developer, may be tempted to use with the Java API:

Interface to concrete class casting
Interface cross-casting
Interface downcasting

It is important to understand that objects returned from methods within ArcObjects can behave differently than objects implicitly defined in your code because the object reference is not held in the JVM.

If you have a method, doSomeProcessingOnPolygon(Polygon p), that operates only on Polygon objects, and you want to pass the object obtained as a result of IFeature.getShape(), you need a way to convert the ‘type’ of the object from IGeometry to Polygon. In Java, this is done using a class cast operation as shown on the slide.

在Java中外接口和事件

- 使用Java监听器可以连接到COM事件
- 添加 <Listener> to 外部接口
- 技术
  - 添加和移除listener类
  - 如果要使用控件必须使用event adaptor class

```
// Wire up the events for MapControl
map.addIMapControlEvents2Listener( new IMapControlEvents2Adapter() {
  public void onAfterDraw( IMapControlEvents2OnAfterDrawEvent evt ) throws IOException {
    if( evt.getViewDrawPhase() == esriViewDrawPhase.esriViewForeground ) {
      try {
        // draw the shape on the MapControl
        map.drawShape( currentExtent, fillSymbol );
     }catch( Exception e )
        { System.err.println( "Error in drawing shape on MapControl" );
     }
   }
}
```

Copyright © 2004 ESRI. All rights reserved.

Events in Java

In Java, events are generated by objects. An event is represented by a java.util.EventObject subclass that carries event information. There are subclasses for each kind of event. At a minimum, they hold a reference to the object that generated the event (the source of the event), but each subclass defines additional information appropriate for the event.

For example, a javax.swing.JButton object generates a java.awt.event.ActionEvent when pressed. It creates an instance of the java.awt.event.ActionEvent class, populates it with information about the event, including a reference to itself, and notifies the interested parties about the event.

Similarly, all ArcObjects-JavaBeans are capable of firing events. For instance, the MapControl bean fires the following events:

OnAfterDraw: Fires after the Map draws a specified view phase.

OnAfterScreenDraw: Fires after the Map contained by the MapControl has finished drawing.

OnBeforeScreenDraw: Fires before the Map contained by the MapControl starts to draw.

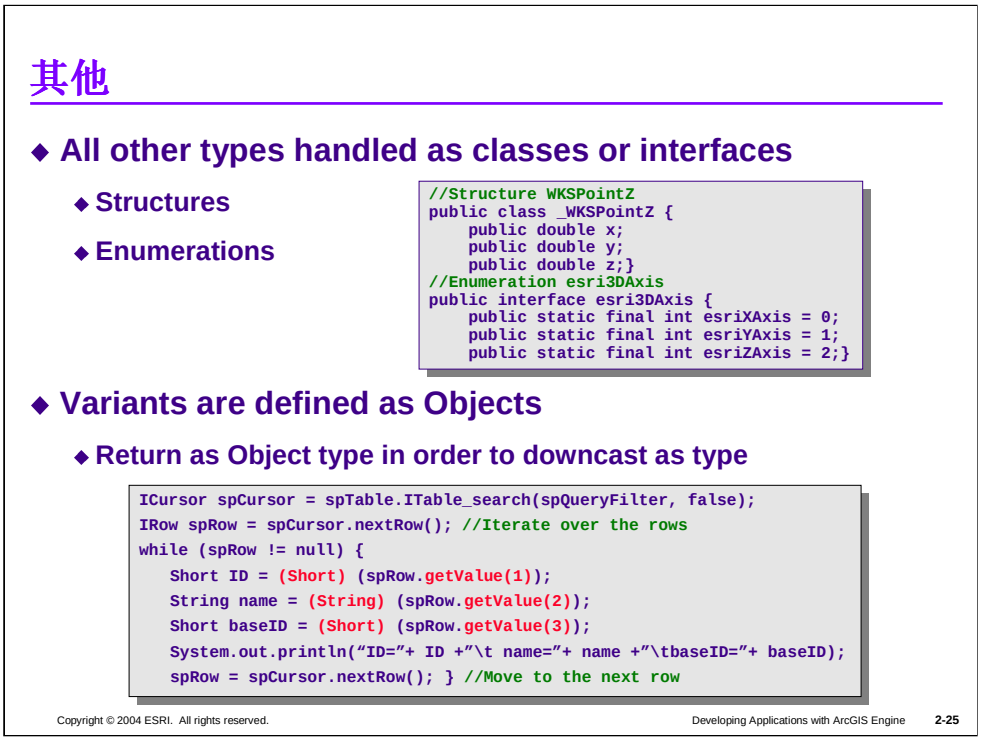
OnDoubleClick: Fires when the user presses and releases the mouse button twice in quick succession.

OnExtentUpdated: Fires after the extent (visible bounds) of the MapControl is changed.

OnFullExtentUpdated: Fires after the full extent (bounds) of the MapControl has changed.

OnKeyDown: Fires after a key is pressed on the keyboard.

OnKeyUp: Fires after a pressed key is released.



Structures

A structure defines a new data type made up of elements called *members*. Java does not have structures as complex data types. The Java language provides this functionality though classes; you can simply declare a class with the appropriate instance variables. For each structure in ArcObjects, there is a representative Java class with publicly declared instance variables matching the structure members as outlined in the slide.

These classes can be referred to like any other class in Java:

```
_WKSPointZ pt = new _WKSPointZ(); pt.x = 2.23; pt.y = -23.14; pt.z = 4.85;
System.out.println(pt.x + " " + pt.y + " " + pt.z);
```

Enumerations

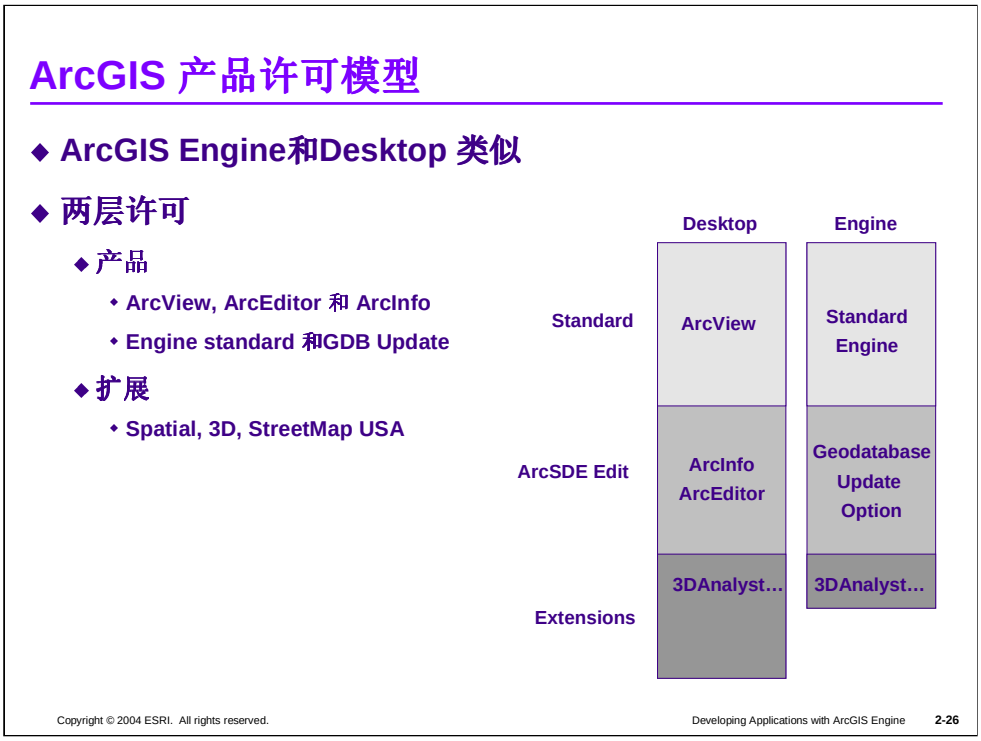
Java 2 SDK prior to version 5 did not have enum types. To emulate enumerations in Java, a class or interface must be created that holds constants. For each enumeration in native ArcObjects, there is a Java interface with publicly declared static integers representing the enumeration value.

The esriXAxis constant can be accessed using the following notation:

```
esri3DAxis.esriXAxis;
```

Variants

The variant data type can contain a wide array of subtypes. With variants, all types can be contained within a single type variant. Everything in the Java programming language is an object. Even primitive data types can be encapsulated inside objects if required. Every class in Java extends java.lang.Object; consequently, methods in ArcObjects that take variants as parameters can be passed any object type in the Java API.



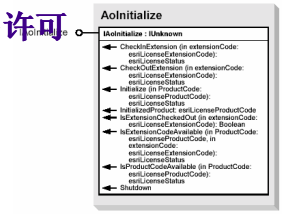
ArcGIS Licensing Model

There are two types of ArcGIS runtime licenses: product licenses and extension licenses. Product licenses include core ESRI software such as ArcGIS Engine Runtime, ArcGIS Desktop licenses (ArcView, ArcEditor, ArcInfo), and ArcGIS Server. ArcGIS Engine Runtime is the platform on which ArcGIS Desktop is built, the ArcGIS Desktop licensing may factor into your application development and is discussed here.

Custom applications that utilize core ArcGIS Engine feature can be built to run against any or all of the following product runtime licenses: ArcGIS Engine Runtime, ArcView, ArcEditor, and ArcInfo. Applications that utilize extended ArcGIS Engine features require extension licenses that correspond to the ArcGIS product providing the core license. In other words, if your application initializes with an ArcGIS Engine Runtime, any needed extension licenses must be ArcGIS Engine Runtime options, not the ArcGIS Desktop extensions.

ArcGIS Engine runtime 许可: 产品

- 在运行时，所有应用程序都必须初始化产品许可
- 使用 esriSystem::AoInitialize初始化产品许可
- 可编写的产品
  - ArcView
  - ArcEditor
  - ArcInfo
  - Engine
  - EngineGeoDB




```
Private m_pAoInitialize As IAoInitialize

Private Sub Form_Load()
  Set m_pAoInitialize = New AoInitialize
  m_pAoInitialize.Initialize esriLicenseProductCodeEngine
End Sub

Private Sub Form_Unload(Cancel As Integer)
  m_pAoInitialize.Shutdown
End Sub
```

Copyright © 2004 ESRI. All rights reserved.

The initialization process

Each stand-alone application developed using ArcObjects must initialize itself with a suitable ArcGIS license to ensure it runs successfully on any machine to which it is deployed. The license initialization must be performed by the application, at application start time, before any ArcObjects components are accessed. Failure to initialize results in application errors.

The AoInitialize coclass must be used by developers to initialize each application with suitable licenses in order for it to run successfully on any machine it is deployed onto. It initializes the application with the specified product license. The product license determines the functionality the application will be able to access. Once the product license has been initialized, it cannot be changed for the duration of the applications life, as it is not possible to re-initialize the application.

Before initializing the application, use the IsProductCodeAvailable and IsExtensionCodeAvailable methods to ensure the appropriate licenses are available to initialize the application with. If all the licenses you require are available using Engine Single Use, then we recommend you use it in preference to the Desktop Concurrent and Desktop Single Use licenses. This means you will not limit the Desktop Concurrent licenses available to any other users.

ArcGIS Engine runtime 许可: 扩展

- 应用程序还可以初始化扩展许可
- 使用AoInitialize管理扩展的许可
- 可编写的扩展
  - 3DAnalyst, SpatialAnalyst, 和StreetMap

```
Private m_pAoInitialize As IAoInitialize

Private Sub Form_Load()
  Set m_pAoInitialize = New AoInitialize
  m_pAoInitialize.Initialize esriLicenseProductCodeEngine
  m_pAoInitialize.CheckOutExtension esriLicenseExtensionCode3DAnalyst
End Sub

Private Sub Form_Unload(Cancel As Integer)
  m_pAoInitialize.CheckInExtension (esriLicenseExtensionCode3DAnalyst)
  m_pAoInitialize.Shutdown
End Sub
```

Copyright © 2004 ESRI. All rights reserved.

Licensing: Extensions

CheckOutExtension::IAoInitialize checks out the specified extension license. Before using the CheckOutExtension method use the IsExtensionCodeAvailable method to ensure the extension license is available with the product used to initialize the application with.

Extensions can either be checked out as and when an application requires the extension functionality, and checked in once the application has finished with the functionality; or the extension can be checked out directly after the application is initialized and checked back in before Shutdown. Use the CheckInExtension method to check extensions back in. The way that the extensions are checked in and out will depend on the type of product license passed to the Initialize method.

If the application was initialized with either of the Engine Single Use licenses, any extensions used by the application will also be Engine Single Use. As such, any extensions can be checked out directly after the application is initialized and checked back in before Shutdown.

Using the CheckOutExtension method to check out an extension in an ArcGIS desktop application will not enable the user interface objects in the extension. Use the IExtension object to access the extension functionality and limit the use of the AoInitialize object to testing the availability of extension licenses and determining the extensions already checked out.

The CheckOutExtension method will fail to check out the specified extension license if:

- The extension is not licensed.
- The license is unavailable because it is already being used. An extension may fail to check Out because it may have been checked out by another application since the IsExtensionCodeAvailable method checked its availability.
- An unexpected license failure occurs due to system administration problems.
- The license is already checked out. Use IsExtensionCheckedOut method to determine this.

区别ArcGIS Engine和 Desktop许可

- 应用程序能初始化ArcGIS Engine或Desktop许可
- 使用IAoInitialize::IsProductCodeAvailable
- 可以使用License来对用户进行选择 …

```
 ' Try to check out Engine license first
If m_pAoInitialize.IsProductCodeAvailable(esriLicenseProductCodeEngine) =
  esriLicenseAvailable Then

  m_pAoInitialize.Initialize esriLicenseProductCodeEngine

' Try to check out Desktop license next
ElseIf
  m_pAoInitialize.IsProductCodeAvailable(esriLicenseProductCodeArcView) =
   esriLicenseAvailable Then

  m_pAoInitialize.Initialize esriLicenseProductCodeArcView

End If
```

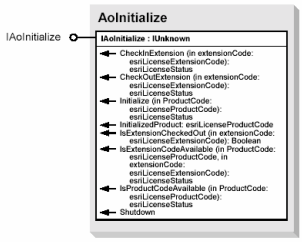
Copyright © 2004 ESRI. All rights reserved.

The IsProductCodeAvailable method returns whether the specified license is available. If the license is available, it can be used to Initialize the application.

If the product you require is not available, you may want to check whether a higher product license is available to initialize the application with. If the license you require is available using Engine Single Use, then we recommend you use it in preference to the Desktop Concurrent and Desktop Single Use licenses. This means you will not limit the Desktop Concurrent licenses available to any other users. If there are no appropriate product licenses available, the application should inform the user of the issue, and either allow the user to resolve the issue or exit the application.

The runtime license 生命周期

- 步骤
  1. **Check if the product is available with the *IsProductCodeAvailable* method.**
  2. **Check what extensions are available with the *IsExtensionCodeAvailable* method.**
  3. ***Initialize* the application with the product license.**
  4. **As required, 执行 extension check outs and check ins with the *CheckOutExtension* and *CheckInExtension* methods.**
  5. ***Shutdown* the product license.**



Copyright © 2004 ESRI. All rights reserved.

The license initialization process in an Engine application

The initialization of an application with a license must be performed in the following order:

1. Check whether the product license is available. The product license that is chosen determines the functionality the application will be able to access. Once the product license has been initialized, it cannot be changed for the duration of the application's life.

2. Check whether the extension licenses are available. If an application has been designed to use extension functionality, it may check for the availability of extension licenses before the application is initialized. Checking the availability of an extension license must be done in conjunction with the product license that the application will ultimately be initialized with, as not every extension license is available with every product license.

3. Initialize the application with the product license. Once it has been established that the appropriate product and extension licenses are available, the application should be initialized with the product license. Once initialized, it is not possible to reinitialize the application.

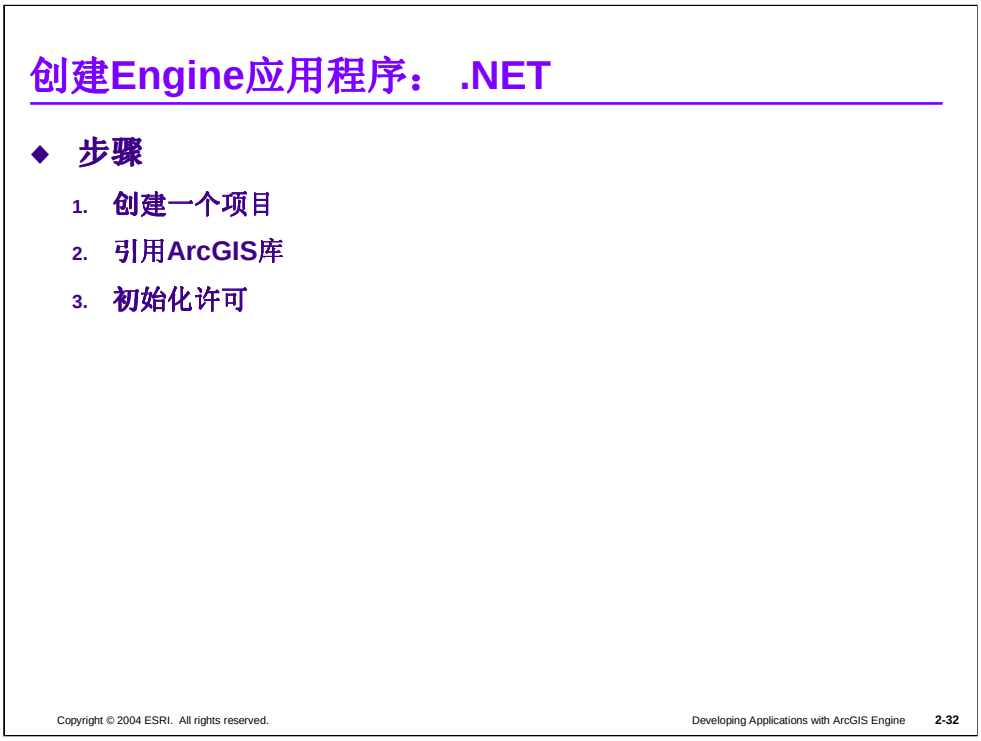
4. As required, perform extension check-outs and check-ins. Extensions can either be checked out when an application requires the extension functionality and checked in once the application has finished with the functionality, or the extension can be checked out directly after the application is initialized and checked back in before shutdown. The way that the extensions are checked in and out will depend on the type of product license with which the application was initialized.

5. Shut down the application. Before an application is shut down, the AoInitialize object must be shut down. This ensures that any ESRI libraries that have been used are unloaded.

Licensing 规则

- 在执行任何ArcObjects code 之前都必须对License initialization
- 在应用程序还在持续的时候，应用程序都将检查 license 是否存在
- 所有扩展 licenses 都要被登记并且是合格的才能使用
- Cannot combine or mix product and extension licenses
- Should write code to query the level of licensing available and 检查Licenses的来源
- 尽量初始化使用中用到的最小化的 licensing

Copyright © 2004 ESRI. All rights reserved.



Creating Engine applications: .NET

In order to create a stand-alone ArcGIS Engine application in .NET, you must follow the following steps.

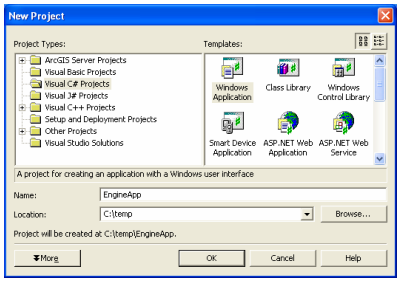
1. Create a project. Create a Windows application project in VB.NET or C# as it is the foundation of most solutions that involve Windows Forms.

2. Reference the ArcGIS libraries. Libraries contain components that expose services used by the other libraries composing ArcGIS. In order to add required ArcGIS libraries, in the Solution Explorer window, right-click References > Add Reference... > .NET. If you do not want to manually add the libraries to your project, you can use the ESRI Automatic references add-in to automatically add ESRI .NET assembly references to the project. This add-in can be downloaded from the ESRI downloads website.

3. Initialize the license.

步骤1:创建一个独立项目

- 创建一个Windows应用程序
  - VB.NET或C#
- 项目名称是一个namespace
  - EngineLicenseApp



Copyright © 2004 ESRI. All rights reserved.

Create a stand-alone project

You must create a Windows Application project in Visual Studio in order to create a stand-alone application using ArcObjects.

In .NET be aware that the default namespace for a project is the project name. Even if you change the default namespace in the project settings, the project name will still be used for the assembly and type library name if the assembly is registered for COM Interop.

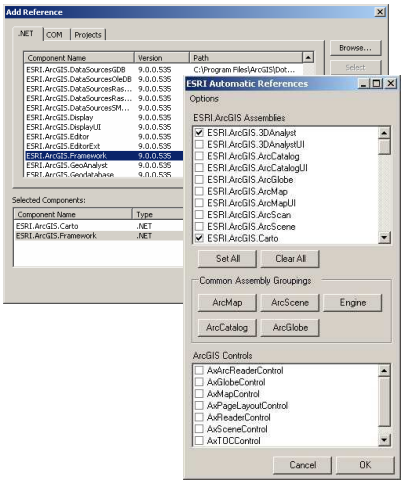
The namespace will often be modified in the project setting. This is to adhere to Microsoft's recommendation of using Company.Technology as the first part of the namespace to help avoid naming conflicts.

步骤2: 引用ArcGIS库

- 手动引用ESR集合引用
- 开发工具
  - ESRI Automatic References add-in
- 在所有类里导入引用

```
Imports ESRI.ArcGIS.esriSystem

Public Class frmEngineClient
    Inherits System.Windows.Forms.Form
```



Copyright © 2004 ESRI. All rights reserved.

Referencing ArcGIS libraries in .NET

When you develop with ArcObjects in .NET you must reference the ESRI .NET interop assemblies (provided with ArcGIS Desktop, Engine, and Server).

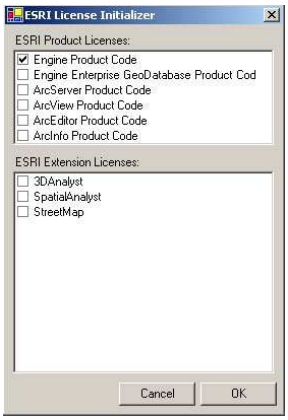
Although .NET cannot use COM objects directly, the COM interoperability services provided by the .NET framework allow you to use COM objects in your .NET project. Using the tools available with Visual Studio .NET, you can create a runtime callable wrapper which allows a .NET component to create and call methods on a COM object. You can also create a COM callable wrapper, which allows a COM client to use .NET components as if they were COM objects. ESRI provides .NET runtime callable wrappers, which will be used to instantiate and make calls on the objects in the ESRI object libraries from your .NET project.

To reference additional class libraries, click the Project menu and click Add Reference. When the Add Reference dialog opens, make sure to select the .NET tab. Almost all projects need a reference to ESRI.ArcGIS.SystemUI since this library contains the plug in interfaces for commands and tools (e.g., ICommand and ITool).

ESRI also provides an add-in that automatically adds ESRI .NET assembly references to a project. The add-in works with one VB.NET or C# project at a time. If the .NET solution contains multiple projects, the add-in works by using the ‘active’ project that contains the document currently visible in either code or design view. To access the add-in, select the ESRI AutomaticReferences menu item on the Tools menu.

步骤3:初始化许可

- 手动实现这段代码
- Developer tools
  - ESRI License Initializer add-in




```
Private m_pAoInitialize As IAoInitialize

Private Sub Form_Load()
  Set m_pAoInitialize = New AoInitialize
  m_pAoInitialize.Initialize esriLicenseProductCodeEngine
End Sub

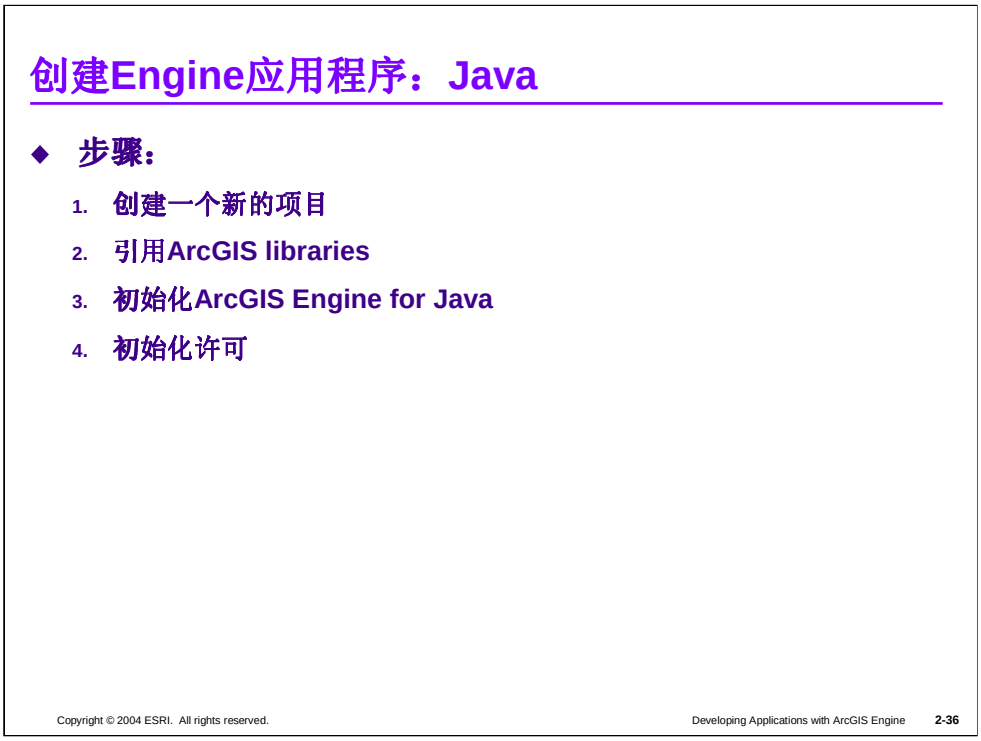
Private Sub Form_Unload(Cancel As Integer)
  m_pAoInitialize.Shutdown
End Sub
```

Copyright © 2004 ESRI. All rights reserved.

License initialization

As mentioned earlier in this lesson, each stand-alone application developed using ArcObjects must be initialized with a suitable ArcGIS license to ensure it runs successfully on any machine which it is deployed to.

You can manually write the license initialization code in your project or you can use the ESRI license initializer add-in tool to automatically generate and add license initialization code to an ArcObject project. The ESRI license initializer add-in tool can be downloaded from the ESRI downloads website and can be used in VB or .NET projects.



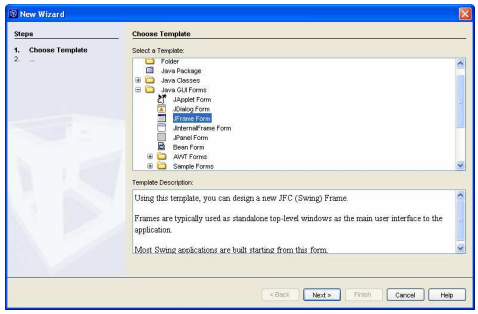
Creating Engine applications: Java

In order to create a stand-alone ArcGIS Engine application in Java, you must follow the following steps:

1. Create a project using one of the supported IDEs (e.g., JBuilder, NetBeans).
2. Reference the ArcGIS libraries in order to gain access to the objects they contain.
3. Initialize ArcGIS Engine for Java using the EngineInitializer class.
4. Initialize the license using the AoInitialize coclass.

步骤1: 创建一个独立项目

- 设置目录位置
  - C:\Student\DAAE
- 选择Jframe窗口
- 设置名字和位置
  - 比如: DaaeEngineClient
- 定义开发包名字
  - 比如: com.esri.arcgis.daae.engine



Copyright © 2004 ESRI. All rights reserved.

Creating a stand-alone project in Java

The detailed steps for creating a project in Java depends on the IDE used. Below are the general steps used in NetBeans 3.6.

1. Create and define the properties for a new project (e.g., choosing the project name and location).
2. Create and define a new “Java Package” (e.g., com.esri.arcgis.daae.engine).
3. Add a new “JFrame Form” to the project and name it.
4. Add controls to the form (e.g., “JButton”).

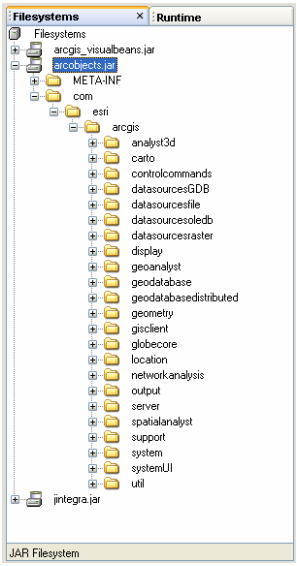
The above steps will be used in the Java version of Exercise 2.

步骤2: 引用ArcGIS libraries

- 激活ESRI 开发包
- 可以访问所有类
- 在所有类中导入引用

```
package com.esri.arcgis.daae.engine;

import javax.swing.*;
import java.io.IOException;
import com.esri.arcgis.system.*;
```



Copyright © 2004 ESRI. All rights reserved.

Mounting ESRI libraries

Since your code will depend on ESRI libraries, you have to mount these libraries in order to add them to the project's Java classpath. As discussed earlier, ESRI libraries are contained in JAR files. You can mount each of these JAR files as a filesystem in the IDE by right-clicking the root filesystem node and choosing Mount > Archive File. Mounting ESRI libraries this way makes all of their contents available for code completion.

Using the import statement

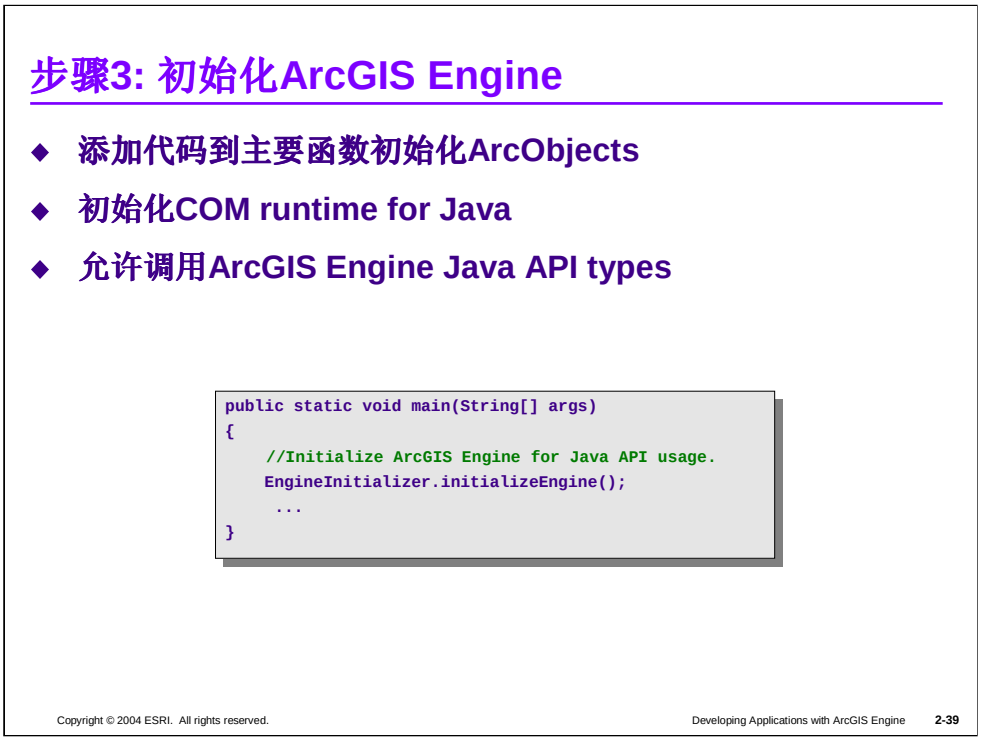
Before you can refer to classes or objects in other packages, you must import the package or the specific entities within the package to which you want to refer.

Import statements must be the first non-comment items in the file after the optional package statement.

To import all classes and objects in a specific package use the .* option (e.g., import com.esri.arcgis.system.*;).

To import only the class or object in the package to which you want to refer, you may refer to it specifically (e.g., import com.esri.arcgis.geodatabase.row;).

Although it might not have significant effects on the code behavior, when importing only a few classes from a large package, the specific form should be used instead of importing the entire package.



The EngineInitializer

In order to initialize your Java environment for native usage of ArcObjects, every ArcGIS Engine Java application must call the static initializeEngine() method on the EngineInitializer class. This should be the first call you do, even before AoInitialize.

The EngineInitializer class is located in the ESRI System library. Besides the initializeEngine() method, the EngineInitializer has several methods most import of which is initializeVisualBeans() which initializes ArcObjects for Java with options optimal for using visual Java Beans in ArcGIS Engine.

步骤4: 初始化许可

- 手动执行代码Manually implement the code
- Developer tools
  - ESRI License Initializer add-in

```
public DaaeEngineClient() {
       initComponents();
       this.show();
       try{
           DisableCommandButtons();
           m_aoInitialize = new AoInitialize();
           m_isProductInitialized = false;
           initArcObjects();
           UpdateStatus();
       }
       catch (Exception ex) {
           ex.printStackTrace();
       }
   }
```

Copyright © 2004 ESRI. All rights reserved.

License initialization

As mentioned earlier in this lesson, each stand-alone application developed using ArcObjects must be initialized with a suitable ArcGIS license to ensure it runs successfully on any machine which it is deployed to.