毕业论文5000字英文翻译怎么找啊

篇一：毕业设计的5000字英文文献翻译

外文及翻译

英语原文 Android Application Fundamentals

Android applications are written in the Java programming language. The Android SDK tools compile the code —along with any data and resource files —into an Android package, an archive file with an .apk suffix. All the code in a single .apk file is considered to be one application and is the file that Android-powered devices use to install the application. Once installed on a device, each Android application lives in its own security sandbox: ? The Android operating system is a multi-user Linux system in which each

application is a different user.

? By default, the system assigns each application a unique Linux user ID (the ID is used only by the system and is unknown to the application). The system sets

permissions for all the files in an application so that only the user ID assigned to that application can access them.

? Each process has its own virtual machine (VM), so an application's code runs in isolation from other applications.

? By default, every application runs in its own Linux process. Android starts the process when any of the application's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other applications.

In this way, the Android system implements the principle of least privilege. That is, each application, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an application cannot access parts of the system for which it is not given permission.

However, there are ways for an application to share data with other applications and for an application to access system services:

? It's possible to arrange for two applications to share the same Linux user ID, in which

case they are able to access each other's files. To conserve system resources,

applications with the same user ID can also arrange to run in the same Linux process

and share the same VM (the applications must also be signed with the same

certificate).

? An application can request permission to access device data such as the user's

contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and

more. All application permissions must be granted by the user at install time.

That covers the basics regarding how an Android application exists within the system. The rest of this document introduces you to: ? The core framework components that define your application.

? The manifest file in which you declare components and required device features for

your application.

? Resources that are separate from the application code and allow your application to

gracefully optimize its behavior for a variety of device configurations.

Application Components

Application components are the essential building blocks of an Android application. Each component is a different point through which the system can enter your application. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role —each one is a unique building block that helps define your application's overall behavior.

There are four different types of application components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

Here are the four types of application components:

Activities

An activity represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email application, each one is independent of the others. As such, a different application can start any one of these

activities (if the email application allows it). For example, a camera application can start the activity in the email application that composes new mail, in order for the user to share a picture.

An activity is implemented as a subclass of Activity and you can learn more about it in the Activities developer guide.

Services

A service is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.

A service is implemented as a subclass of Service and you can learn more about it in the Services developer guide.
Content providers

A content provider manages a shared set of application data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your application can access. Through the content provider, other applications can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any application with the proper permissions can query part of the content provider (such as ContactsContract.Data) to read and write information about a particular person.

Content providers are also useful for reading and writing data that is private to your application and not shared. For example, the Note Pad sample application uses a content provider to save notes.

A content provider is implemented as a subclass of ContentProvider and must implement a standard set of APIs that enable other applications to perform transactions. For more information, see the Content Providers developer guide.

Broadcast receivers

A broadcast receiver is a component that responds to system-wide broadcast aements. Many broadcasts originate from the system —for example, a broadcast aing that the screen has turned off, the battery is low, or a picture was captured. Applications can also initiate broadcasts —for example, to let other applications know that some data has been downloaded to the device and is available for them to use.

Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More ly, though, a broadcast

intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.

A broadcast receiver is implemented as a subclass of BroadcastReceiver and each broadcast is delivered as an Intent object. For more information, see theBroadcastReceiver class.

A unique aspect of the Android system design is that any application can start another application's comp onent. For example, if you want the user to capture a photo with the device camera, there's probably another application that does that and your application can use it, instead of developing an activity to capture a photo yourself. You don't need to incorporate or even link to the code from the camera application. Instead, you can simply start the activity in the camera application that captures a photo. When complete, the photo is even returned to your application so you can use it. To the user, it seems as if the camera is actually a part of your application.

When the system starts a component, it starts the process for that application (if it's not already running) and instantiates the classes needed for the component. For example, if your application starts the activity in the camera application that captures a photo, that activity runs in the process that belongs to the camera application, not in your application's process.

Therefore, unlike applications on most other systems, Android applications don't have a single entry point (there's no main() function, for example).

Because the system runs each application in a separate process with file permissions that restrict access to other applications, your application cannot directly activate a component from another application. The Android system, however, can. So, to activate a component in

another application, you must deliver a message to the system that specifies your intent to start a particular component. The system then activates the component for you.

Activating Components

Three of the four component types —activities, services, and broadcast receivers —are activated by an asynchronous message called an intent. Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your application or another.

An intent is created with an Intent object, which defines a message to activate either a specific component or a specific type of component —an intent can be either explicit or implicit, respectively.

For activities and services, an intent defines the action may specify the URI of the data to act on (among other things that the component being started might need to know). For example, an intent might convey a request for an activity to show an image or to open a web page. In some cases, you can start an activity to receive a result, in which case, the activity also returns the result in

an Intent (for example, you can issue an intent to let the user pick a personal contact and have it returned to you—the return intent includes a URI pointing to the chosen contact).

For broadcast receivers, the intent simply defines the aement being broadcast (for example, a broadcast to indicate the device battery is low includes only a known action string

The other component type, content provider, is not activated by intents. Rather, it is

activated when targeted by a request from a ContentResolver. The content resolver handles all direct transactions with the content provider so that the component that's performing

transactions with the provider doesn't need to and instead calls methods on

the ContentResolver object. This leaves a layer of abstraction between the content provider and the component requesting information (for security).

There are separate methods for activating each type of component:

篇二：毕业论文5000字英文文献翻译(c++)

┆┆┆┆┆┆┆┆┆┆┆┆┆装┆┆┆┆┆订┆┆┆┆┆线┆┆┆┆┆┆┆┆┆┆┆┆┆

安徽工业大学 毕业设计（论文）说明书

英文翻译

英语原文：

. Introducing Classes

The only remaining feature we need to understand before solving our bookstore problem is how to write a data structure to represent our transaction data. In C++ we define our own data structure by defining a class. The class mechanism is one of the most important features in C++. In fact, a primary focus of the design of C++ is to make it possible to define class types that behave as naturally as the built-in types themselves. The library types that we've seen already, such as istream and ostream, are all defined as classesthat is,they are not strictly speaking part of the language.

Complete understanding of the class mechanism requires mastering a lot of information. Fortunately, it is possible to use a class that someone else has written without knowing how to define a class ourselves. In this section, we'll describe a simple class that we canuse in solving our bookstore problem. We'll implement this class in the subsequent chapters as we learn more about types,expressions, classes.

To use a class we need to know three things:What is its name? Where is it defined?

What operations does it support?

For our bookstore problem, we'll assume that the class is named Sales_item and that it is defined in a header named Sales_item.h.The Sales_item Class

The purpose of the Sales_item class is to store an ISBN and keep track of the number of copies sold, the revenue, and average sales price for that book. How these data are stored or computed is not our concern. To use a class, we need not know anything about how it is implemented. Instead, what we need to know is what operations the class provides.

As we've seen, when we use library facilities such as IO, we must include the associated headers. Similarly, for our own classes, we must make the definitions associated with the class available to the compiler. We do so in much the same way. Typically, we put the class definition into a file. Any program that wants to use our class must include that file.

Conventionally, class types are stored in a file with a name that, like the name of a program source file, has two parts: a file name and a file suffix. Usually the file name is the same as the class defined in the header. The suffix usually is .h, but some programmers use .H, .hpp, or .hxx. Compilers usually aren't picky about header file names, but IDEs sometimes are. We'll assume that our class is defined in a file named Sales_item.h.Operations on Sales_item Objects

Every class defines a type. The type name is the same as the name of the class. Hence, our Sales_item class defines a type named

Sales_item. As with the built-in types, we can define a

are saying that item is an object of type Sales_item. We

In addition to being able to define variables of type Sales_item, we can perform the following operations on Sales_item objects:

Use the addition operator, +, to add two Sales_items,Use the input operator, << to read a Sales_item object,Use the output operator, >> to write a Sales_item object,

Use the assignment operator, =, to assign one Sales_item object to another,

Call the same_isbn function to determine if two Sales_items refer to the same book.Classes are central to most C++ programs: Classes let us define our own types that are customizedfor the problems we need to solve, resulting in applications that are easier to write and understand.Welltypes.A class defines data and function members: The data members store the state associated with objectsof the class type, and the functions perform operations that give meaning to the data. Classeslet us separate implementation and interface. The interface specifies the operations that the classsupports. Only the implementor of the class need know or care about the details of the implementation. This separation reduces the bookkeeping aspects that make programming tedious anderror-prone.

Class types often are referred to as abstract data types. An abstract data type treats the data(state) and operations on that state as a single unit. We can think abstractly about what the classd oes, rather than always having to be aware of how the class operates. Abstract data types arefundamental to both object-oriented and generic programming.

Data abstraction is a programming (and design) technique that relies on the separation of interfaceand implementation. The class designer must worry about how a class is implemented, but programmersthat use the class need not know about these details. Instead, programmers who use a type need to know only the type's interface; they can think abstractly about what the type does rather than concretely about how the type works.

Encapsulation is a term that describes the technique of combining lower-level elements to forma new, higher-level entity. A function is one form of encapsulation: The detailed actions performedby the function are encapsulated in the