1. Service onStartCommand 返回值：START\_STICKY、START\_NOT\_STICKY、START\_REDELIVER\_INTENT、START\_STICKY\_COMPATIBILITY
   1. */\*\*  
       \* Constant to return from {****@link*** *#onStartCommand}: compatibility  
       \* version of {****@link*** *#START\_STICKY} that does not guarantee that  
       \* {****@link*** *#onStartCommand} will be called again after being killed.*

*\*/***public static final int *START\_STICKY\_COMPATIBILITY*** = 0;  
  
*/\*\*  
 \* Constant to return from {****@link*** *#onStartCommand}: if this service's  
 \* process is killed while it is started (after returning from  
 \* {****@link*** *#onStartCommand}), then leave it in the started state but  
 \* don't retain this delivered intent. Later the system will try to  
 \* re-create the service. Because it is in the started state, it will  
 \* guarantee to call {****@link*** *#onStartCommand} after creating the new  
 \* service instance; if there are not any pending start commands to be  
 \* delivered to the service, it will be called with a null intent  
 \* object, so you must take care to check for this.  
 \*   
 \* <p>This mode makes sense for things that will be explicitly started  
 \* and stopped to run for arbitrary periods of time, such as a service  
 \* performing background music playback.  
 \*/***public static final int *START\_STICKY*** = 1;  
  
*/\*\*  
 \* Constant to return from {****@link*** *#onStartCommand}: if this service's  
 \* process is killed while it is started (after returning from  
 \* {****@link*** *#onStartCommand}), and there are no new start intents to  
 \* deliver to it, then take the service out of the started state and  
 \* don't recreate until a future explicit call to  
 \* {****@link*** *Context#startService Context.startService(Intent)}. The  
 \* service will not receive a {****@link*** *#onStartCommand(Intent, int, int)}  
 \* call with a null Intent because it will not be re-started if there  
 \* are no pending Intents to deliver.  
 \*   
 \* <p>This mode makes sense for things that want to do some work as a  
 \* result of being started, but can be stopped when under memory pressure  
 \* and will explicit start themselves again later to do more work. An  
 \* example of such a service would be one that polls for data from  
 \* a server: it could schedule an alarm to poll every N minutes by having  
 \* the alarm start its service. When its {****@link*** *#onStartCommand} is  
 \* called from the alarm, it schedules a new alarm for N minutes later,  
 \* and spawns a thread to do its networking. If its process is killed  
 \* while doing that check, the service will not be restarted until the  
 \* alarm goes off.  
 \*/***public static final int *START\_NOT\_STICKY*** = 2;  
  
*/\*\*  
 \* Constant to return from {****@link*** *#onStartCommand}: if this service's  
 \* process is killed while it is started (after returning from  
 \* {****@link*** *#onStartCommand}), then it will be scheduled for a restart  
 \* and the last delivered Intent re-delivered to it again via  
 \* {****@link*** *#onStartCommand}. This Intent will remain scheduled for  
 \* redelivery until the service calls {****@link*** *#stopSelf(int)} with the  
 \* start ID provided to {****@link*** *#onStartCommand}. The  
 \* service will not receive a {****@link*** *#onStartCommand(Intent, int, int)}  
 \* call with a null Intent because it will will only be re-started if  
 \* it is not finished processing all Intents sent to it (and any such  
 \* pending events will be delivered at the point of restart).  
 \*/***public static final int *START\_REDELIVER\_INTENT*** = 3;

* 1. START\_NOT\_STICKY
     1. 如果系统在onStartCommand()方法返回之后杀死这个服务，那么