Программный комплекс

Универсальный телекоммуникационный сервер для построения распределенных систем мониторинга и управления на основе технологий облачных вычислений (ПК УТС «Облако»)

Руководство пользователя

2018

СОДЕРЖАНИЕ

1 Состав

Программный комплекс Универсальный телекоммуникационный сервер для построения распределенных систем мониторинга и управления на основе технологий облачных вычислений - ПК УТС «Облако» (далее Сервер) состоит из следующих основных модулей:

- основной исполняемый файл запуска сервера;
- дополнительные модули и файлы их настроек;

2 Назначение

Программный комплекс Универсальный телекоммуникационный сервер для построения распределенных систем мониторинга и управления на основе технологий облачных вычислений - ПК УТС «ОБЛАКО» (далее Сервер) предназначен для построения распределенных систем мониторинга и управления на основе технологий облачных вычислений.

Программный комплекс Универсальный телекоммуникационный сервер для построения распределенных систем мониторинга и управления на основе технологий облачных вычислений - ПК УТС «ОБЛАКО» это комплекс сервисных программ и модулей, доступных разработчику. Сервер предоставляет интегрированную среду для развертывания и выполнения высокопроизводительных серверных бизнес-приложений и управления ими. Эти приложения могут обслуживать запросы, принимаемые от удаленных клиентских систем, в том числе, подключающихся из Интернета, корпоративной сети или интрасети. Предоставляет разработчикам упрощенную модель программирования сетевых серверных приложений. Разработчики могут использовать встроенные библиотеки для реализации в приложениях множества функций, таких как ввод-вывод, обработка численных данных и текста, доступ к базам данных, обработка XML-кода, управление транзакциями, бизнес-правила и веб-службы.

3 Запуск и остановка Сервера

Существует два способа запуска Сервера:

- Ручной;
- Служба Windows.

При ручном способе запуска настраивается ярлык, и запуск Сервера происходит по двойному клику по ярлыку. Если Сервер запускается в качестве службы Windows, то запуск происходит вместе с загрузкой Windows. Как настроить нужный тип запуска, изложено в Инструкции по установке и запуску.

4 Остановка Сервера

4.1 Остановка Сервера, запущенного вручную

1) Окно «Incom.Server.Starter.exe - Ярлык» сделать активным.
2) Нажать «Enter».

Все запущенные сервисы будут последовательно остановлены, после чего окно «Incom.Server.Starter.exe - Ярлык» закроется.

4.2 Остановка Сервера, запущенного в качестве службы Windows

1) Открыть список служб.
2) Выделить службу «Incom Application Server» и нажать «Остановить».
3) Дождаться остановки службы.

4.3 Удаление из списка служб Windows

1) Запустить системную консоль командной строки от имени администратора.
2) Перейти в каталог, где установлен Сервер, набрав команду:
   cd C:\Program Files\ss_uts_cloud
3) выполнить команду: incom.server.starter.exe –u.
4) Открыть список служб и убедиться, что служба «Incom Application Server» отсутствует в списке.

5 Дополнительные параметры запуска сервера

Дополнительные параметры Сервера определяются в файле Incom.Server.config в корневой папке сервера. Во время загрузки ядра, Сервер проверяет наличие этого файла. Если файл найден параметры из него загружаются и регистрируются в RC.

Предполагается что параметры - это статичные параметры сервера, которые не меняются в ходе рабы. И перезапись этого файла не осуществляется.

Пример:

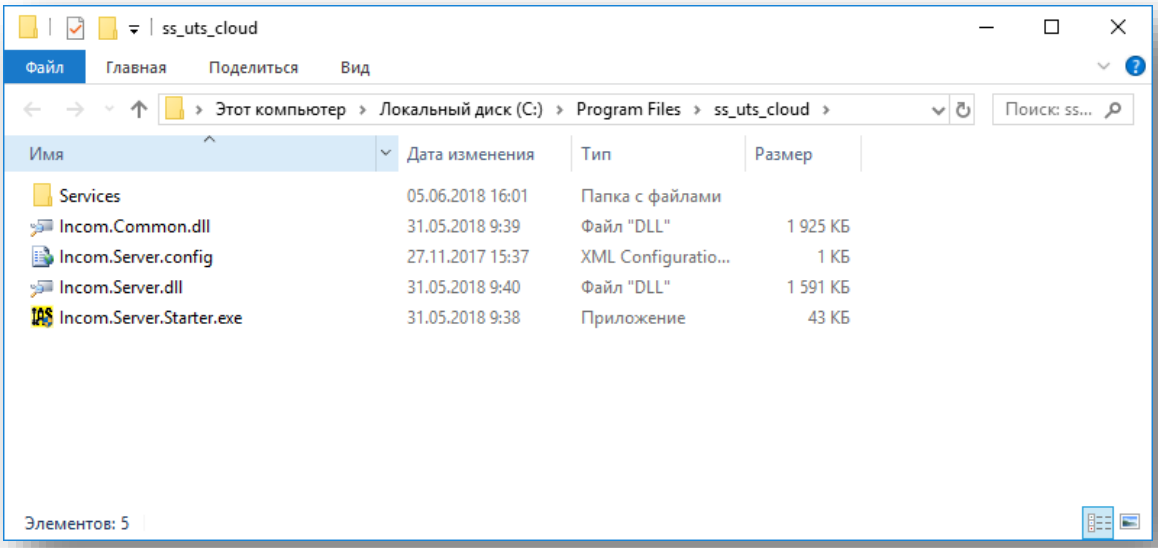
```
<configuration>
      <server name="IncomApplicationServer"><parameter name="DisplayName"
value="Incom Application Server" />
            <parameter name="LogName" value="IncomApplicationServerLog" />
            <parameter name="ConfigEncoding" value="UTF8" />
            <parameter name="ConfigPersistenceType" value="OnStopNoBuf" />
            <parameter name="FileSystemBlockSize" value="4096" />
            <parameter name="ConfigRepairOnStart" value="True" />
            <parameter name="Parameter1" value="..." />
            <parameter name="Parameter2" value="..." />
      </server>
</configuration>
```

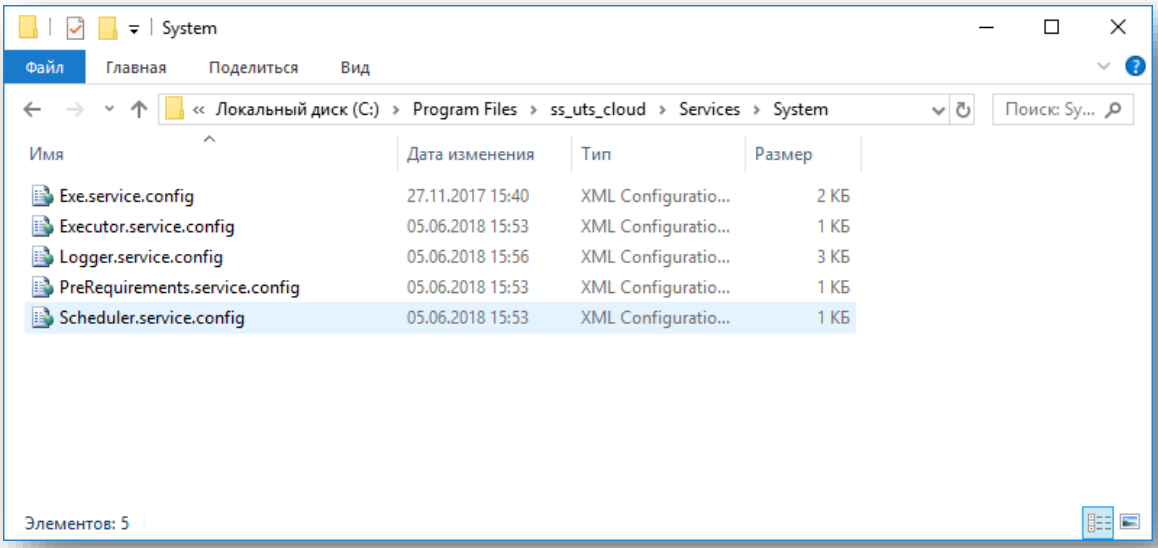
- LogName – имя ситемного журнала сообщений.
- ConfigEncoding – Кодировка для конфигураций сервисов IAS. Отсутствие данного параметра равносильно заданию UTF8. Данная кодировка считается основной для Сервера по умолчанию. При сохранении конфигураций сервер форсирует замену кодировок в xml файлах на данную.
- ConfigPersistenceType¶ - Параметр отвечает за тип сохранения конфигурации сервисов перезапись конфигурации сервера во время остановки. Варианты:
    - ConfigPersistenceType = OnStop - перезапись осуществляется во время остановки сервера
    - ConfigPersistenceType = OnStopNoBuf - перезапись осуществляется во время остановки сервера, для записи используется WinApi и выравнивание файлов на FileSystemBlockSize
    - ConfigPersistenceType = OnRequest - перезапись осуществляется только по явному запросу через management
    - ConfigPersistenceType = OnRequestNoBuf - перезапись осуществляется только по явному запросу через management, для

записи используется WinApi и выравнивание файлов на FileSystemBlockSize
    - значение не задано - перезапись осуществляется только по явному запросу через management
- FileSystemBlockSize¶ -размер кластера на диске. Необходимый параметр для режимов записи в обход буффера используя WinApi
- ConfigRepairOnStart¶ - метка восстанавливать битые конфигурации дополнительных модулей на старте. Бекапные версии конфигурации дополнительных модулей появляются после правильной остановки сервера. И содержат версию конфигов с которыми сервер успешно запустился последний раз. Если размер конфигурационного файла оказывается меньше 6 байт (размер заголовка xml), файл считается испорченным и восстанавливается из папки с Backup если задан ключ. Также восстановление происходит если xml парсер не может загрузить текущую версию конфигурационного файла и падает с исключением XmlException. Перед восстановлением конфигурационного файла выполняется сохранение испорченного конфигурационного файла в папке Backup с добавлением расширения .bad

6 Дополнительные модули

Дополнительные модули (библиотеки *.dll) и их конфигурации (файлы *.service.config) распологаются в подпапках папки Services.





Пример конфигурации:

Scheduler.service.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <service name="SchedulerService" assemblyName="Incom.Server"
      className="Incom.Server.Services.Scheduling.SchedulerService">
    <parameter name="MaxTaskCount" value="1000" />
    <parameter name="ExecutorServiceName" value="ExecutorService" />
    <parameter name="WaitTimeout" value="1000" />
    <dependOn name="LoggerService" />
    <dependOn name="ExecutorService" />
  </service>
</configuration>
```

Один файл конфигурации может содержать много запускаемых дополнительных модулей (далее Сервисов).

Базовый класс для всех Сервисов Сервера - ServiceBasis наследуется от MarshalByRefObject.

Сервис наследуемый от ServiceBasis или AsynchronousServiceBasis должен задать своё описание в конструкторе и вызвать базовый конструктор.

Также он должен реализовать методы:

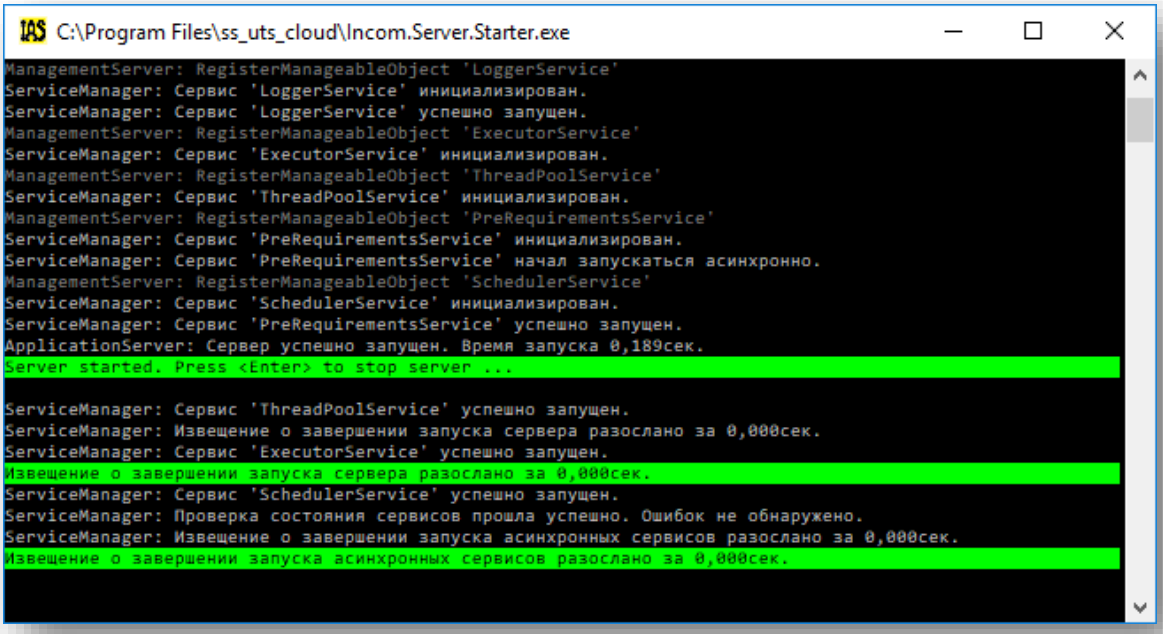
- InitializeService - метод инициализация сервиса, должен подготовить сервис к запуску, сервер не гарантирует порядок инициализации сервисов.
- StartService - метод запуска сервиса, сервер гарантирует порядок запуска сервисов в соответствии с зависимостями.
- StopService - метод остановки сервиса.
- UnInitializeService - метод деинициализации сервиса.
- Также он может перегружать методы:
- ServerStarted - Извещение о корректном запуске синхронных сервисов сервера. Запрещается долгая обработка данного вызова.
- ServerAllAsincServicesStarted - Извещение о завершении запуска всех асинхронных сервисов на сервере. Запрещается долгая обработка данного вызова.
- PrepareStop - Подготавливает к остановке. В данном методе можно актуализировать состояние параметров сервиса перед остановкой.

Доступные свойства сервиса:

- Name - Название сервиса.
- Description - Описание сервиса.
- State - Состояние сервиса. Может быть: Initialized, Started, Stopped, UnInitialized.

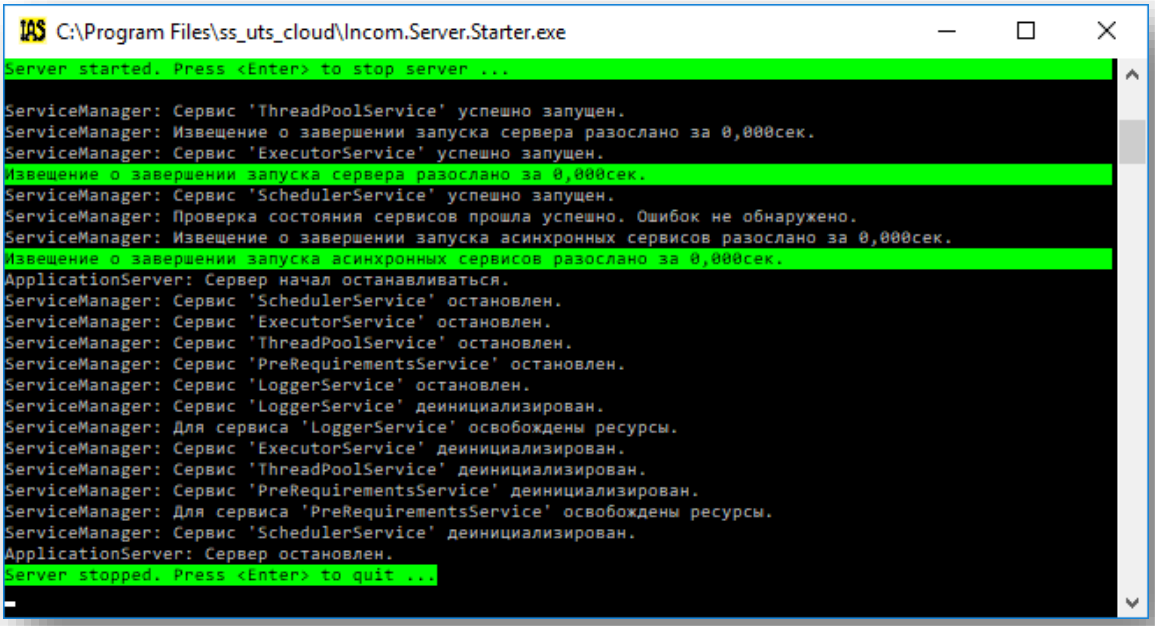
Во время запуска сервер:

1. находит конфигурацию и создаёт экземпляр сервиса в ней описанный
2. инициализирует параметры сервиса
3. выполняет инициализацию сервиса ('InitializeService()') и анализирует зависимости
4. запускает сервис ('InitializeService(StartService)') (только после того как все сервисы от которых зависит данные сервис запущены). Если сервис наследуется от AsynchronousServiceBasis запуск происходит в асинхронном потоке. Это также приводит к тому что если от данного сервиса кто то зависел он тоже будет запущен асинхронно.
5. Когда все синхронные сервиса запущены, сервер проводит оповещение всех сервисов используя ServerStarted.
6. Когда все асинхронные сервиса запущены, сервер проводит оповещение всех сервисов используя ServerAllAsincServicesStarted .
7. Когда все синхронные и асинхронные сервиса запущены, сервер проводит диагностику и выводит предупреждения о не стартовавших сервисах.



Во время остановки сервер:

1. сервер проводит оповещение всех сервисов используя PrepareStop.
2. останавливает сервисы в обратном порядке в соответствии с зависимостями
3. деинициализирует сервисы



Пример реализации сервиса:

Пример

```
/// <summary>
/// Мой сервис.
/// </summary>
public class MyService
    : ServiceBasis
{
    /// <summary>
    /// Конструктор по умолчанию
    /// </summary>
    public MyService() : base()
    {
        description = "Сервис журналирования.";
    }
    /// <summary>
    /// Параметр сервиса.
    /// </summary>
    public int MyParameter
    {
        get;
        set;
    }
    /// <summary>
    /// Инициализирует сервис
    /// </summary>
    protected override void InitializeService()
    {
        //
    }
    /// <summary>
    /// Запускает сервис
    /// </summary>
    protected override void StartService()
    {
        //
    }
    /// <summary>
    /// Извещение о корректном запуске сервера.
    /// </summary>
    protected override void ServerStarted()
    {
            log.Info(String.Format("Я {0}. Дошли слухи что IAS запустился.", this.Name));
    }
    /// <summary>
    /// Останавливает сервис
    /// </summary
    protected override void StopService()
    {
        //
    }
    /// <summary>
    /// Деинициализирует сервис
    /// </summary>
    protected override void UnInitializeService()
    {
        //
    }
}
```