Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Отчет по производственной практике

Выполнил: Руководитель от вуза:

Студент группы 153501 Летохо А. С.

Иванов И.И. Руководитель от предприятия:

Петров Г. Р.

Минск 2016

Оглавление

[Введение 2](#_Toc423100859)

[Структура предприятия 3](#_Toc423100860)

[Общие сведения о предприятии 3](#_Toc423100861)

[Организационная структура компании 3](#_Toc423100862)

[Основные методы защиты информации на предприятии 5](#_Toc423100863)

[Обзор инфраструктуры информационных технологий предприятия 5](#_Toc423100864)

[Методы и средства защиты информации на предприятии 5](#_Toc423100865)

[Обзор используемых технологий разработки платформы Java 7](#_Toc423100866)

[Общие сведения о платформе Java 7](#_Toc423100867)

[Использование платформы Java на предприятии 8](#_Toc423100868)

[Java Swing 8](#_Toc423100869)

[Java RMI 10](#_Toc423100870)

[JAXB 11](#_Toc423100871)

[Обзор подходов к автозапуску приложений в ОС семейства Linux 13](#_Toc423100872)

[Общие сведения о Linux 13](#_Toc423100873)

[Автозапуск в Linux 14](#_Toc423100874)

[Автозапуск программы как демона 15](#_Toc423100875)

[Реализация автозапуска Java приложения в среде Linux в виде демона 17](#_Toc423100876)

[**ЗАКЛЮЧЕНИЕ** 20](#_Toc423100877)

[Приложение 21](#_Toc423100878)

# Введение

ООО «Югис Групп» является высокоэффективной ИТ — компанией, предоставляющей ИТ-услуги как на внутреннем рынке Республики Беларусь, так и за рубежом.

Политика компании направленная на постоянное улучшение качества предоставляемых услуг и повышение компетенции технического персонала позволило пополнить портфолио компании успешно реализованными проектам по внедрению систем мониторинга для сотовых операторов Республики Беларусь, сопряжению аппаратных датчиков интенсивности дорожного трафика с централизованной автоматизированной системой управления дорожным движением города Сочи, внедрению системы сервис-деск в национальном центре электронных услуг.

Руководство ООО «Югис Групп» держит руку на пульсе рынка технологий и на данный момент компания активно наращивает техническую компетенцию в области машинного обучения.

# Структура предприятия

## Общие сведения о предприятии

Компания ООО «ЮГИС Групп» была основана в 2012 году и являлась компанией - системным интегратором, предлагающей решения корпоративного уровня на базе программных продуктов HP.

На сегодняшний день основными направлениями деятельности компании являются:

* Внедрение проприетарных систем сервис-деск, а так же сервис-деск систем, распространяющихся свободно;
* ИТ – консалтинг в области управления ИТ – инфраструктурой предприятия, бизнес-проектирование на базе сборника лучших практик ITIL;
* Внедрение систем мониторинга ИТ – инфраструктуры предприятия;
* Системное интегрирование разрозненных инфраструктурных и бизнес–систем в единую инфраструктуру предприятия;
* Разработка программного обеспечения на заказ.

На сегодняшний день компания «ЮГИС Групп» входит в холдинг компаний ООО «Пиксел инвестментс».

## Организационная структура компании

Согласно схеме (рис. 1) предприятие ООО «ЮГИС Групп» состоит из руководящих органов, группы производственных отделов и одного непроизводственного. Ниже приведено описание основных отделов предприятия.

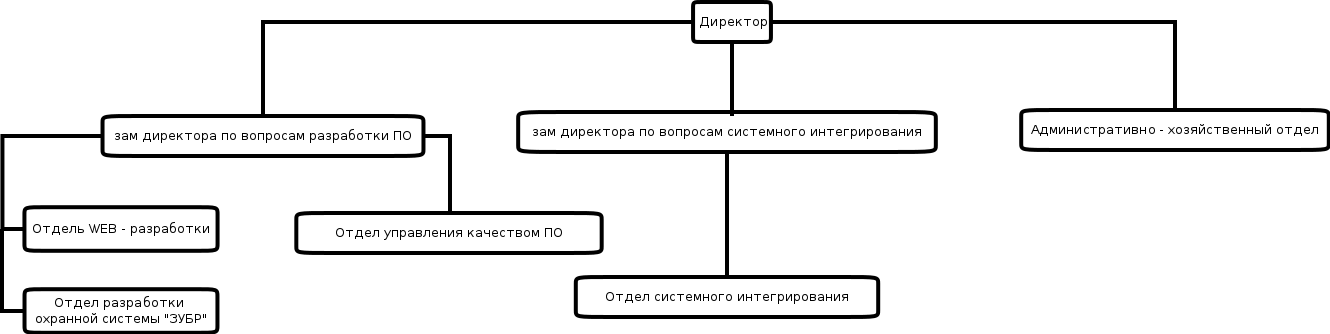


Рисунок 1 — организационная структура ООО «ЮГИС Групп»

К руководящим органам предприятия относятся следующие:

1 Директор – высшее должностное лицо компании. В задачу входит определение общей стратегии компании, принятие решений на высшем уровне, выполнение представительских обязанностей.

2 Зам директора по вопросам разработки ПО – технический директор. Является одним из руководителей компании, выполняет функции управления проектам, связанными с разработкой ПО

3 Зам директора по вопросам системного интегрирования – технический директор. Является одним из руководителей компании, выполняет функции управления проектами, связанными с системным интегрированием, внедрением систем мониторинга и внедрением систем сервис-деск.

Производственными отделами предприятия являются:

1 Отдел Web-разработки - разработка легковесных в сети интернет.

2 Отдел разработки системы охраны периметра «ЗУБР» - разработка программной части аппаратно-программного комплекса периметровой сигнализации «ЗУБР».

3 Отдел управления качеством ПО — функциональное и исследовательское тестирование ПО, автоматизированное ПО, помощь отделам разработки во внедрении практик модульного тестирования и непрерывной интеграции.

4 Отдел системного интегрирования — внедрение и интеграция в ИТ-инфраструктуру предприятия сервис-деск систем, систем мониторинга ИТ –инфраструктуры, система автоматической инвентаризации сети, администрирование ИТ – инфраструктуры ООО «ЮГИС Групп» и т. д.

Непроизводственным отделом предприятия является отдел административно — хозяйственной деятельности (АХО). Отдел АХО отвечает за бухгалтерский учёт, кадровое делопроизводство, снабжение офиса

# Основные методы защиты информации на предприятии

## Обзор инфраструктуры информационных технологий предприятия

Инфраструктура информационных технологий предприятия (ИТ – инфраструктура) состоит из локальной вычислительной сети, расположенной в офисе организации и вычислительного облака, расположенного на арендуемых площадях центра обработки данных.

На базе вычислительных мощностей, расположенных в ЦОД развёрнуты критические ИТ-сервисы компании, такие как система управления проектами (Redmine), система управления тестовыми сценариями (TestCase DB), сервис-деск система и другие.

Между офисом компании и центром обработки данных существует два канала связи: один основной, предоставляющий доступ к серверам напрямую через сеть провайдера, второй, запасной, работающий через сеть интернет.

## Методы и средства защиты информации на предприятии

Для обеспечения конфиденциальности информации предприятия весь сетевой трафик между офисом компании и центром обработки передаётся через шифрованный VPN — туннель, построенный на базе технологии OpenVPN.

Все внутренние и внешние сервисы компании используют защищенный протокол аутентификации пользователей KERBEROS 5, а так же центральный сервис аутентификации пользователей на базе стека технологии Microsoft Active Directory.

Внешние ИТ-сервисы компании доступные закачзиком через сеть интернет, такие как система учёта ошибок, портал самообслуживания для заказчиков компании, расположены в демилитаризованной зоне (DMZ, построенное по схеме с одним межсетевым экраном (рис 2).

Для обеспечения высокого уровня доступности ИТ-инфраструктуры, все критические ИТ-сервисы охвачены круглосуточным мониторингом на базе HP SiteScope и HP Business Service Management.

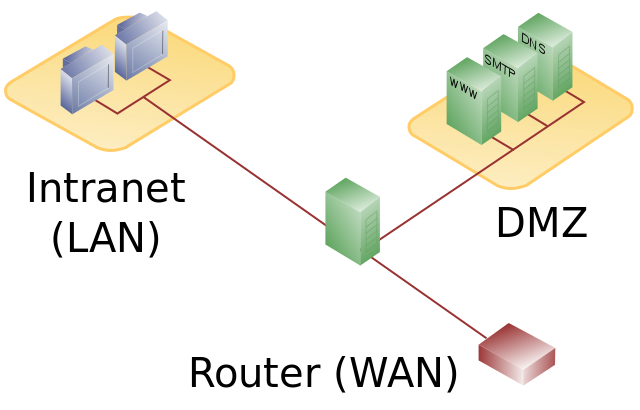


Рисунок 2 — Схема DMZ с одним NAT

Наряду с техническими средставами защиты информации на предприятии используются организационные методы защиыт информации. Например, для обеспечения целостности и доступности информации для всех информаицонных систем регулярно снимаются резервные копии данных. Для каждой системы хранятся эталонный снимок рабочего окружения, полученный с помощью функции VMWare Snapshoot Virtual Machine, а так же план восстановления системы после сбоя.

План восстановления системы после сбоя представляет собой пошаговую инструкцию восстановления системы в случае возникновения внештатной ситуации.

Для обеспечения целостности данных в центре обработки данных используется система хранения данных HP D2200sb, работающая в режиме RAID5.

# Обзор используемых технологий разработки платформы Java

## Общие сведения о платформе Java

**Программная платформа Java** — ряд [программных продуктов](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) и спецификаций компании [Sun Microsystems](https://ru.wikipedia.org/wiki/Sun_Microsystems), ранее независимой компании, а ныне дочерней компании корпорации [Oracle](https://ru.wikipedia.org/wiki/Oracle), которые совместно предоставляют систему для разработки [прикладного программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) и встраивания её в любое [кросс-платформенное программное обеспечение](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). [Java](https://ru.wikipedia.org/wiki/Java) используется в самых разных [компьютерных платформах](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0) от [встраиваемых устройств](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D1%82%D1%80%D0%B0%D0%B8%D0%B2%D0%B0%D0%B5%D0%BC%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) и [мобильных телефонов](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%82%D0%B5%D0%BB%D0%B5%D1%84%D0%BE%D0%BD) в нижнем ценовом сегменте, до корпоративных серверов и суперкомпьютеров в высшем ценовом сегменте. Хотя [Java-апплеты](https://ru.wikipedia.org/wiki/Java-%D0%B0%D0%BF%D0%BF%D0%BB%D0%B5%D1%82) редко используются в [настольных компьютерах](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D1%82%D0%BE%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), однако они в них иногда используются для улучшения функциональности и повышения безопасности при просмотре [всемирной паутины](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D0%B0).

Программный код, написанный на Java, [виртуальная машина Java](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_Java) преобразует в [байт-код Java](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4_Java). Однако есть компиляторы [байт-кода](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4) для других языков программирования, таких как [Ada](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%B0_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [JavaScript](https://ru.wikipedia.org/wiki/JavaScript), [Python](https://ru.wikipedia.org/wiki/Python), и [Ruby](https://ru.wikipedia.org/wiki/Ruby). Также есть несколько новых языков программирования, разработанных для работы с виртуальной машиной Java. Это такие языки как [Scala](https://ru.wikipedia.org/wiki/Scala_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [Clojure](https://ru.wikipedia.org/wiki/Clojure) and [Groovy](https://ru.wikipedia.org/wiki/Groovy). [Синтаксис Java](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81_Java&action=edit&redlink=1) в основном заимствован из [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) и [C++](https://ru.wikipedia.org/wiki/C%2B%2B), но объектно-ориентированные возможности основаны на модели, используемой в [Smalltalk](https://ru.wikipedia.org/wiki/Smalltalk) и [Objective-C](https://ru.wikipedia.org/wiki/Objective-C). В Java отсутствуют определённые [низкоуровневые](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) конструкции, такие как [указатели](https://ru.wikipedia.org/wiki/%D0%A3%D0%BA%D0%B0%D0%B7%D0%B0%D1%82%D0%B5%D0%BB%D1%8C_(%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)), также Java имеет очень простую модель памяти, где каждый объект расположен [в куче](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D0%B8) и все переменные [объектного типа](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF&action=edit&redlink=1) являются [ссылками](https://ru.wikipedia.org/wiki/%D0%A1%D1%81%D1%8B%D0%BB%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). Управление памятью осуществляется с помощью интегрированной автоматической [сборки мусора](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), которую выполняет [JVM](https://ru.wikipedia.org/wiki/JVM).

[13 ноября](https://ru.wikipedia.org/wiki/13_%D0%BD%D0%BE%D1%8F%D0%B1%D1%80%D1%8F) [2006 года](https://ru.wikipedia.org/wiki/2006_%D0%B3%D0%BE%D0%B4) компания Sun Microsystems сделала большую часть своей реализации Java доступной в соответствии с [GNU General Public License](https://ru.wikipedia.org/wiki/GNU_General_Public_License) (GPL), хотя некоторые части поставляются в скомпилированном виде из-за вопросов авторского права с кодом, на который имеет лицензию (но не право собственности) компания Sun Microsystems.

Программная платформа Java — это имя для [пакета программ](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D0%BA%D0%B5%D1%82_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC) компании Sun, которые позволяют разрабатывать и запускать программы, написанные на языке программирования Java. Эта программная платформа не является специфической для какого-либо одного [процессора](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) или [операционной системы](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), но механизм выполнения (называемый [виртуальной машиной](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0)) и [компилятор](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) с набором библиотек, которые реализованы для различного [аппаратного обеспечения](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) и различных операционных систем, чтобы Java-программы могли работать везде одинаково.

* [Java Card](https://ru.wikipedia.org/wiki/Java_Card): Технология, которая позволяет небольшим Java-приложениям ([апплетам](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D0%BB%D0%B5%D1%82)) надежно работать на [смарт-картах](https://ru.wikipedia.org/wiki/%D0%A1%D0%BC%D0%B0%D1%80%D1%82-%D0%BA%D0%B0%D1%80%D1%82%D0%B0) и других подобных устройств c малым объёмом [памяти](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C).
* [Java ME](https://ru.wikipedia.org/wiki/Java_Platform,_Micro_Edition): Включает в себя несколько различных наборов библиотек (известных как профили) для устройств с ограниченным объёмом места для хранения, небольшим размером [дисплея](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BF%D0%BB%D0%B5%D0%B9) и [батареи](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D1%82%D0%B0%D1%80%D0%B5%D1%8F_(%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0)). Часто используется для разработки приложений для мобильных устройств, [КПК](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D0%BC%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), [ресиверов цифрового телевидения](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%81%D0%B8%D0%B2%D0%B5%D1%80_%D1%86%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%BE%D0%B3%D0%BE_%D1%82%D0%B5%D0%BB%D0%B5%D0%B2%D0%B8%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F) и [принтеров](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%82%D0%B5%D1%80).
* [Java SE](https://ru.wikipedia.org/wiki/Java_Platform,_Standard_Edition): для использования на настольных [ПК](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), [серверах](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%B0%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)) и другом подобном оборудовании.
* [Java EE](https://ru.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition): Java SE плюс [API](https://ru.wikipedia.org/wiki/API), полезное для [многоуровневых](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0&action=edit&redlink=1) ([*англ.*](https://en.wikipedia.org/wiki/Multitier_architecture)) [клиент-серверных](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80) [бизнес-приложений](https://ru.wikipedia.org/w/index.php?title=%D0%91%D0%B8%D0%B7%D0%BD%D0%B5%D1%81-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F&action=edit&redlink=1) ([*англ.*](https://en.wikipedia.org/wiki/Enterprise_software)).

## Использование платформы Java на предприятии

В компании платформа Java используется отделом разработки системы охраны периметра «ЗУБР» для построения серверной и клиентской части программного обеспечения системы.

В построении клиент-серверной части системы охраны используются следующие технологии платформы Java:

* Java SE
* Java Swing
* Java RMI
* JAXB

### Java Swing

**Swing** — [библиотека](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) для создания [графического интерфейса](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) для программ на языке [Java](https://ru.wikipedia.org/wiki/Java). Swing был разработан компанией [Sun Microsystems](https://ru.wikipedia.org/wiki/Sun_Microsystems). Он содержит ряд графических [компонентов](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81%D0%B0) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Swing widgets*), таких как кнопки, поля ввода, таблицы и т. д.

Swing относится к библиотеке классов [JFC](https://ru.wikipedia.org/wiki/Java_Foundation_Classes), которая представляет собой набор библиотек для разработки графических оболочек. К этим библиотекам относятся [Java 2D](https://ru.wikipedia.org/w/index.php?title=Java_2D&action=edit&redlink=1), [Accessibility-API](https://ru.wikipedia.org/w/index.php?title=Accessibility-API&action=edit&redlink=1), [Drag & Drop](https://ru.wikipedia.org/wiki/Drag_%26_Drop)-API и [AWT](https://ru.wikipedia.org/wiki/Abstract_Window_Toolkit).

[Архитектура](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) Swing разработана таким образом, что вы можете изменять «[look and feel](https://ru.wikipedia.org/w/index.php?title=Look_and_feel&action=edit&redlink=1" \o "Look and feel (страница отсутствует))» (L&F) вашего приложения. «Look» определяет внешний вид компонентов, а «Feel» — их поведение. Sun’s JRE предоставляет следующие L&F:

* *CrossPlatformLookAndFeel* — это родной L&F для Java-приложений (так же называется Metal). Он используется по умолчанию, обеспечивая стандартное поведение компонентов и их внешний вид, вне зависимости от платформы, на которой запускается приложение.
* *SystemLookAndFeel* — в этом случае приложение использует L&F, который является родным для системы, на которой запущено приложение. Системный L&F определяется во время выполнения. Для Windows используется «Windows» L&F, который имитирует особенности конкретной системы, на которой запущен — классический Windows, XP, или Vista. Для Linux и Solaris используется «GTK+», если установлен GTK+ 2.2 или более поздняя версия, в противном случае используется «Motif».
* *Synth* — основа для создания собственных L&F.
* *Multiplexing* — предоставляет возможность использования различных L&F одновременно.
* Swing предоставляет более гибкие интерфейсные компоненты, чем более ранняя библиотека [AWT](https://ru.wikipedia.org/wiki/Abstract_Window_Toolkit). В отличие от AWT, компоненты Swing разработаны для одинаковой [кросс-платформенной](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) работы, в то время как компоненты AWT повторяют интерфейс исполняемой [платформы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B0) без изменений. AWT же использует только стандартные элементы ОС для отображения, то есть для каждого элемента создается отдельный объект ОС (окно), в связи с чем, AWT не позволяет создавать элементы произвольной формы (возможно использовать только прямоугольные компоненты), элементы управления на основе AWT всегда отображаются поверх Swing-элементов (так как все Swing компоненты отображаются на поверхности контейнера).
* Компоненты Swing поддерживают специфические динамически [подключаемые виды и поведения](https://ru.wikipedia.org/w/index.php?title=Look_and_Feel&action=edit&redlink=1) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *plugable look-and-feel*), благодаря которому возможна адаптация к графическому интерфейсу платформы (то есть к компоненту можно динамически подключить другой, специфический для операционной системы, в том числе и созданный программистом вид и поведение). Таким образом, приложения, использующие Swing, могут выглядеть как родные приложения для данной операционной системы. Основным минусом таких «легковесных» ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Lightweight*) компонентов является относительно медленная работа. Положительная сторона — универсальность интерфейса созданных приложений на всех платформах.

**Принцип Lightweight** -- «Lightweight» означает, что компоненты Swing отрисовываются самими компонентами на поверхности родительского окна, без использования компонентов [операционной системы](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0). В отличие от «Тяжелых» компонентов AWT, в приложении Swing может иметься только одно окно, и все прочие компоненты отрисовываются на ближайшем родителе, имеющем собственное окно (например, на JFrame). В приложении могут сочетаться Swing и AWT элементы, хотя это может порождать некоторые проблемы — в частности, компоненты AWT всегда перекрывают Swing элементы, а также закрывают собой всплывающие меню JPopupMenu и JComboBox. Для предотвращения этого, у этих компонентов имеются методы setLightWeightPopupEnabled(boolean), позволяющие запретить использование «Легковесных» всплывающих элементов. При установке свойства в true (setLightWeightPopupEnabled(true)), AWT элементы не будут перекрывать меню.

### Java RMI

**RMI** (*Remote Method Invocation*) — [программный интерфейс](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) [вызова удаленных методов](https://ru.wikipedia.org/wiki/Remote_Procedure_Call) в языке [Java](https://ru.wikipedia.org/wiki/Java), распределенная [объектная модель](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), специфицирующая, каким образом производится вызов удаленных методов, работающих на другой [виртуальной машине Java](https://ru.wikipedia.org/wiki/JVM).

При доступе к объектам на другом компьютере возможно вызывать методы этого объекта. Необходимо только передать параметры метода на другой компьютер, сообщить объекту о необходимости выполнения метода, а затем получить обратно возвращаемое значение. Механизм RMI дает возможность организовать выполнение всех этих операций.

[https://upload.wikimedia.org/wikipedia/commons/thumb/b/ba/RMI-Stubs-Skeletons.svg/400px-RMI-Stubs-Skeletons.svg.png](https://commons.wikimedia.org/wiki/File:RMI-Stubs-Skeletons.svg?uselang=ru)

Типичная реализация модели Java-RMI, использующая объекты 'заглушки'(stub) и 'скелета'(skeleton).

В терминах RMI объект, который вызывает удаленный метод, называется [клиентским объектом](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), а удаленный объект — [серверным объектом](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)). Компьютеры выступают в роли клиента и сервера только для конкретного вызова. Вполне возможно, что при выполнении следующей операции эти компьютеры поменяются ролями, то есть сервер предыдущего вызова может сам стать клиентом при обращении к объекту на другом компьютере.

При вызове метода удаленного объекта на самом деле вызывается обычный метод языка Java, инкапсулированный в специальном объекте-заглушке (stub), который является представителем серверного объекта. Заглушка находится на клиентском компьютере, а не на сервере. Она упаковывает параметры удаленного метода в блок байтов. Каждый параметр кодируется с помощью алгоритма, обеспечивающего независимость от аппаратуры. Например, числа всегда передаются в порядке, при котором сначала передается старший байт (big-endian). При этом объекты подвергаются [сериализации](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B8%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F). Процесс кодирования параметров называется [развертыванием параметров](https://ru.wikipedia.org/w/index.php?title=%D0%A0%D0%B0%D0%B7%D0%B2%D0%B5%D1%80%D1%82%D1%8B%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D0%BE%D0%B2&action=edit&redlink=1) ([parameter marshaling](https://ru.wikipedia.org/w/index.php?title=Parameter_marshaling&action=edit&redlink=1" \o "Parameter marshaling (страница отсутствует))). Основная цель развертывания параметров — преобразование их в формат, пригодный для передачи параметров от одной виртуальной машины к другой.

Метод, принадлежащий заглушке, создает блок, в который входят следующие элементы:

* идентификатор удаленного объекта;
* описание вызываемого метода;
* развернутые параметры.

Затем метод заглушки посылает эту информацию серверу. Далее объект-получатель, скелет (skeleton), выполняет для каждого вызова удаленного метода следующие действия:

* свертывание параметров;
* поиск вызванного объекта;
* вызов заданного метода;
* извлечение и развертывание возвращаемого значения или исключения, сгенерированного данным методом;
* передача пакета, состоящего из развернутых возвращаемых данных, объекту-заглушке на клиентском компьютере.

Клиентский объект-заглушка свертывает возвращаемое значение или исключение, полученное с сервера. Результат свертывания становится возвращаемым значением метода заглушки. Если удаленный метод возвращает исключение, то объект-заглушка повторит его в среде объекта-клиента.

### JAXB

**Java Architecture for XML Binding** (**JAXB**) позволяет [Java](https://ru.wikipedia.org/wiki/Java_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) разработчикам ставить в соответствие Java [классы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и [XML](https://ru.wikipedia.org/wiki/XML) представления. JAXB предоставляет две основные возможности: [*маршаллирование*](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B8%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) Java [объектов](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) в XML и наоборот, то есть *демаршализация* из XML обратно в Java объект. Другими словами, JAXB позволяет хранить и извлекать данные в памяти в любом XML-формате, без необходимости выполнения определенного набора процедур загрузки и сохранения XML. Он похож на xsd.exe и XmlSerializer в [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework).

JAXB особенно полезен, когда спецификация является сложной и меняющейся. В этом случае, постоянные изменения [схемы XML](https://ru.wikipedia.org/wiki/XML_Schema) определений для синхронизации их с определениями Java могут занять много времени и быть подвержены ошибкам.

JAXB является частью [Java SE](https://ru.wikipedia.org/wiki/Java_SE) платформы и является одной из [API](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) в [Java EE платформе](https://ru.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition), и является частью [Java Web Services Development Pack](https://ru.wikipedia.org/w/index.php?title=Java_Web_Services_Development_Pack&action=edit&redlink=1) (JWSDP). Он также является базисом для [WSIT](https://ru.wikipedia.org/w/index.php?title=Web_Services_Interoperability_Technology&action=edit&redlink=1). JAXB является частью SE версии 1.6.

JAXB 1.0 был разработан в рамках [Java Community Process](https://ru.wikipedia.org/wiki/Java_Community_Process) как JSR 31. Как и в 2006, JAXB 2.0 разрабатывается в рамках JSR 222. [Эталонная реализация](https://ru.wikipedia.org/wiki/%D0%AD%D1%82%D0%B0%D0%BB%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%80%D0%B5%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) для этих спецификаций доступна как [CDDL open source license](https://ru.wikipedia.org/wiki/CDDL) на [java.net](https://ru.wikipedia.org/w/index.php?title=Java.net&action=edit&redlink=1).

# Обзор подходов к автозапуску приложений в ОС семейства Linux

## Общие сведения о Linux

**Linux**, также **Ли́нукс** — общее название [Unix-подобных операционных систем](https://ru.wikipedia.org/wiki/Unix-%D0%BF%D0%BE%D0%B4%D0%BE%D0%B1%D0%BD%D0%B0%D1%8F_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), основанных на одноимённом [ядре](https://ru.wikipedia.org/wiki/Linux_(%D1%8F%D0%B4%D1%80%D0%BE)). Ядро Linux создаётся и распространяется в соответствии с моделью разработки [свободного и открытого программного обеспечения](https://ru.wikipedia.org/wiki/FOSS). Поэтому общее название не подразумевает какой-либо единой «официальной» комплектации Linux; они распространяются в основном бесплатно в виде различных готовых [дистрибутивов](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D0%B8%D0%B2_Linux), имеющих свой набор [прикладных программ](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) и уже настроенных под конкретные нужды пользователя.

На начальном этапе Linux бесплатно разрабатывался только энтузиастами-[добровольцами](https://ru.wikipedia.org/wiki/%D0%92%D0%BE%D0%BB%D0%BE%D0%BD%D1%82%D1%91%D1%80%D1%81%D1%82%D0%B2%D0%BE" \o "Волонтёрство), но с успехом Linux и его массовым коммерческим использованием дорабатывать ОС и вносить свой вклад стали и компании, со временем став значительной силой. Подавляющее большинство ПО в современных дистрибутивах по-прежнему доступно по [свободным лицензиям](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D1%8B%D0%B5_%D0%BB%D0%B8%D1%86%D0%B5%D0%BD%D0%B7%D0%B8%D0%B8), как правило, за исключением небольшого количества проприетарных компонентов. В 2008 году расчёты показывали, что для того, чтобы «с нуля» разработать систему, аналогичную [Fedora](https://ru.wikipedia.org/wiki/Fedora) 9, потребовалось бы затратить 10,8 млрд долл.Совокупная себестоимость ядра Linux оценена в более чем 1 млрд [евро](https://ru.wikipedia.org/wiki/%D0%95%D0%B2%D1%80%D0%BE) (около 1,4 млрд долл.). Только за 2008 год себестоимость ядра Linux увеличилась на 225 млн евро. В системе Linux воплощён труд в эквиваленте 73 тыс. [человеко-лет](https://ru.wikipedia.org/wiki/%D0%A7%D0%B5%D0%BB%D0%BE%D0%B2%D0%B5%D0%BA%D0%BE-%D0%B3%D0%BE%D0%B4).

В настоящее время системы Linux лидируют на рынках [смартфонов](https://ru.wikipedia.org/wiki/%D0%A1%D0%BC%D0%B0%D1%80%D1%82%D1%84%D0%BE%D0%BD) ([Android](https://ru.wikipedia.org/wiki/Android" \o "Android) занимает 85 % рынка), [интернет-серверов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%B0%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)) (60 %), самых мощных [суперкомпьютеров](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%BF%D0%B5%D1%80%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) ([97 %](https://ru.wikipedia.org/wiki/TOP500)), а также, согласно [Linux Foundation](https://ru.wikipedia.org/wiki/Linux_Foundation), в [дата-центрах](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D1%82%D0%B0-%D1%86%D0%B5%D0%BD%D1%82%D1%80) и на предприятиях, занимают половину рынка [встраиваемых систем](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D1%82%D1%80%D0%B0%D0%B8%D0%B2%D0%B0%D0%B5%D0%BC%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), имеют значительную долю рынка [нетбуков](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D1%82%D0%B1%D1%83%D0%BA) (32 % на 2009 год). На рынке домашних компьютеров Linux прочно занимает 3 место (по разным данным, от 1 до 5 %). Согласно исследованию [Goldman Sachs](https://ru.wikipedia.org/wiki/Goldman_Sachs), в целом, рыночная доля Linux среди электронных устройств составляет около 42 %.

С тех пор, как ядро Linux было создано для [x86](https://ru.wikipedia.org/wiki/X86)-ПК, оно было [портировано](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) на множество платформ, включая [x86-64](https://ru.wikipedia.org/wiki/X86-64), [PowerPC](https://ru.wikipedia.org/wiki/PowerPC) и [ARM](https://ru.wikipedia.org/wiki/ARM_(%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0)). Linux работает в [роутерах](https://ru.wikipedia.org/wiki/%D0%A0%D0%BE%D1%83%D1%82%D0%B5%D1%80), [телевизорах](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BB%D0%B5%D0%B2%D0%B8%D0%B7%D0%BE%D1%80) и [игровых приставках](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0). ОС на ядре продолжают быстро совершенствоваться (например, новая версия ядра выпускается каждые 2—3 месяца, с 2005 года в разработке ядра принимают участие более 7800 разработчиков из более чем 800 различных компаний) и набирает популярность (за 9 месяцев с мая 2011 по январь 2012 доля Linux выросла на 64 %).

Наиболее популярными дистрибутивами являются (расположены в алфавитном порядке названия их пакетных форматов): [deb](https://ru.wikipedia.org/wiki/Deb_(%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82_%D1%84%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2))-based ([Debian](https://ru.wikipedia.org/wiki/Debian" \o "Debian), [Mint](https://ru.wikipedia.org/wiki/Linux_Mint), [Ubuntu](https://ru.wikipedia.org/wiki/Ubuntu)), [pacman](https://ru.wikipedia.org/wiki/Pacman_(%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BF%D0%B0%D0%BA%D0%B5%D1%82%D0%B0%D0%BC%D0%B8))-based ([Arch Linux](https://ru.wikipedia.org/wiki/Arch_Linux" \o "Arch Linux), [Chakra](https://ru.wikipedia.org/wiki/The_Chakra_Project), [Manjaro](https://ru.wikipedia.org/wiki/Manjaro_Linux)), [RPM](https://ru.wikipedia.org/wiki/RPM)-based ([RedHat](https://ru.wikipedia.org/wiki/RedHat" \o "RedHat), [Fedora](https://ru.wikipedia.org/wiki/Fedora), [Mageia](https://ru.wikipedia.org/wiki/Mageia), [OpenSUSE](https://ru.wikipedia.org/wiki/OpenSUSE)), source-based ([Slackware](https://ru.wikipedia.org/wiki/Slackware" \o "Slackware), [Gentoo](https://ru.wikipedia.org/wiki/Gentoo)).

## Автозапуск в Linux

Существуют несколько способов сделать автозапуск программ в Linux:

* записать вызов программы/скрипта запуска в /etc/rc.local в фоновом режиме (&) (в разных дистрибутивах может лежать в разных местах, например, /etc/rc.d/rc.local) с перенаправленными потоками ввода/вывода в /dev/null. Например, "/home/user/my\_prog 1 > /dev/null 2 > /dev/null &". Также, дополнительно, можно воспользоваться командой [nohup](http://ru.wikipedia.org/wiki/Nohup);
* внести вызов в /etc/inittab, согласно правилам его оформления. В отличие от первого способа тут можно указать [уровень запуска](http://ru.wikipedia.org/wiki/%D0%A3%D1%80%D0%BE%D0%B2%D0%BD%D0%B8_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F) для программы;
* написать скрипт, позволяющий запускать/останавливать/перезапускать программу как демона, а также получать информацию о её состоянии.

Первый способ самый лёгкий, но и самый проблемный. Файл rc.local есть не во всех дистрибутивах. В нём нельзя задать уровень запуска. Если там записано несколько программ, то сложно ими управлять как сервисами (разве что запустить или остановить все одновременно). И, под конец, запуск из него подрывает устойчивость системы от взлома.

Второй метод довольно экзотичный, тем не менее, используя его, нельзя оперировать запущенными таким способом программами как демонами, что довольно неудобно. Да и загромождать inittab некрасиво.

Последний метод на текущий момент самый интересный, но немного сложнее предыдущих (возможно, на первый взгляд). Именно им представлены все системные демоны, что говорит само за себя. Потому его и рассмотрю ниже.

В данном конкретном случае использование первого варианта не подходит, так как требуется независимость от конкретного дистрибутива ОС Linux.

Выбирая между вторым и третьим вариантами было решено, что есть необходимость реализации запуска приложения как демона с возможностью управления им после старта системы. Таким образом было решено реализовывать третий вариант – реализация скрипта запуска в виде демона, лежащего в /etc/init.d.

## Автозапуск программы как демона

Для начала стоит заглянуть в каталог /etc/init.d. Здесь содержатся запускные скрипты всех сервисов, а также два файла для желающих написать себе такой же:  
  
README и skeleton  
  
skeleton содержит в себе болванку скрипта запуска с довольно подробными комментариями, а README его неплохо дополняет, не смотря на его небольшой размер. Также можно посмотреть и другие файлы и попытаться найти там что-то, что прояснит непонятную ситуацию.  
  
В тестовой и целевой системах для запускных скриптов демонов используется LSB (Linux Script Base) Init Standart.   
  
Рассмотрим поближе файл skeleton. Первое с чего он должен начинаться, конечно же "#!/bin/sh", т.к. init-скрипт — запускной файл. Далее идёт комментированный заголовок:

*### BEGIN INIT INFO*

*# Provides: skeleton*

*# Required-Start: $remote\_fs $syslog*

*# Required-Stop: $remote\_fs $syslog*

*# Default-Start: 2 3 4 5*

*# Default-Stop: 0 1 6*

*# Short-Description: Example initscript*

*# Description: This file should be used to construct scripts to be*

*# placed in /etc/init.d.*

*### END INIT INFO*

Может показаться, что это просто лишняя информация от автора, но это не так. То, что указано здесь используется при прописывании скрипта в систему. Тут как раз пригодится файл README, который показывает, что в заголовке skeleton перечислены не все возможные параметры. Как минимум есть ещё следующие:

*# Should-Start: $portmap*

*# Should-Stop: $portmap*

*# X-Start-Before: nis*

*# X-Stop-After: nis*

*# X-Interactive: true*

Все параметры и их полное описание (на английском) можно увидеть на ресурсах:

* [*http://wiki.debian.org/LSBInitScripts*](http://wiki.debian.org/LSBInitScripts) *(англ)*
* [*http://varlog.voan.ru/2009/12/autostart-initd-in-debian.html*](http://varlog.voan.ru/2009/12/autostart-initd-in-debian.html) *(рус)*

К русскому варианту стоит добавить, что в Required-Start: можно прописать $all, тогда текущий скрипт будет запускаться после всех остальных (иногда это бывает нужно). Также X-Interactive: true показывает, что этот скрипт может взаимодействовать с пользователем, запросом на ввод чего-нибудь, например, пароля.  
  
Далее в skeleton идёт инициализация переменных, используемых в самом скрипте. Часть из них нужно будет настроить под свои нужды. Потом проверки на то, что сам демон существует и попытка прочитать конфигурационный файл (их имена должны быть указаны в переменных выше), далее загрузка переменных rcS, а потом идёт одна из самых интересных частей init-файла:  
 *. /lib/lsb/init-functions*  
Это определение LSB функций работы с логами, LSB-статусом сервиса, работы с процессом. В некоторых дистрибутивах этот файл может находиться в каталоге /etc/init.d. Названия и часть подробностей можно узнать непосредственно из комментариев к функциям в этом файле.   
  
Следующая часть — непосредственно тело скрипта. Тело состоит из условных частей, которые являются командами для демона: start, stop, restart/reload/force-reload, status. Кто-то выделяет их в отдельные функции, кто-то нет. На мой взгляд, функциями они выглядят эстетичнее и код более понятен. Все эти команды объединяет оператор выбора case, который и выбирает для исполнения нужный кусок кода, в зависимости от команды (параметра) с которой был запущен init-скрипт.

Таким образом для создания обычного скрипта достаточно подставить в переменные в начале файла нужные значения и, возможно, немного добавить кода в функции start/stop (например загрузку/выгрузку драйвера).  
После того как файл будет готов его нужно скопировать в /etc/init.d и добавить в автозагрузку:  
 *update-rc.d <имя\_скрипта> defaults*  
Удалить из автозагрузки можно так:  
 *update-rc.d -f <имя\_скрипта> remove*  
  
Далее также можно использовать команды sysv-rc-conf в debian или service в fedora core, чтобы включить/выключить автозагрузку сервиса.

# Реализация автозапуска Java приложения в среде Linux в виде демона

Для автозапуска Java приложения необходимо написать скрипт, который будет запускать его в виде системного демона. Для этого воспользуемся заглушкой, представленной в файле /etc/init.d/skeleton

Шаблон, представленный в данном файле предлагает нам использовать start-stop-daemon утилиту, однако она доступна не в каждом дистрибутиве, поэтому придется реализовывать запуск самостоятельно.

Из дополнительных задач, которые необходимо реализовать – обеспечении единственности запущенного экземпляра приложения в системе и возможность остановки приложения по требованию.

Для запуска Java приложения нам необходимо узнать, куда в системе установлен исполняемый пакет java либо получить ссылку на него. После этого запустить Java приложение можно будет командой:

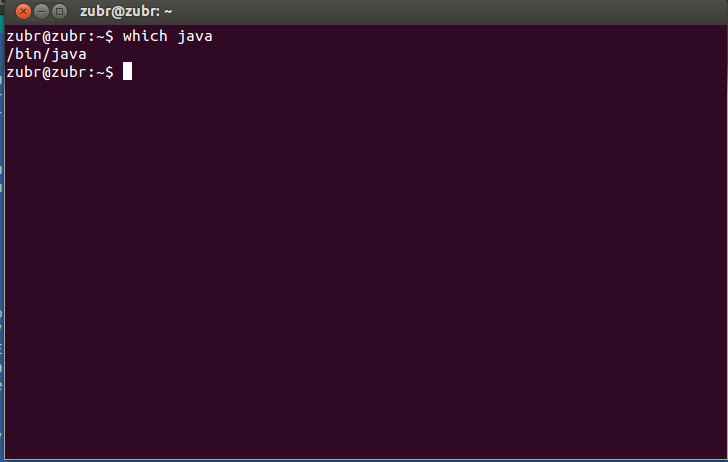
java -jar <путь к jar файлу >

Для обеспечения уникальности запущенного экземпляра воспользуемся PID-файлами, которые будут создаваться при старте приложения и удаляться после его остановки. Однако, мы не можем гарантировать, что приложение всегда будет завершаться «правильно». Вполне возможна ситуация, когда приложение будет «убито» по идентификатору процесса командой kill либо командой killall. В таком случае мы получим ситуацию, когда у нас PID файл есть, а соответствующего ему процесса на самом деле нет. Для обхода этого момента необходимо сделать проверку на присутствие соответствующего PID файлу процесса в системе и, в случае отсутствия, удаления неактуального файла, блокирующего запуск.

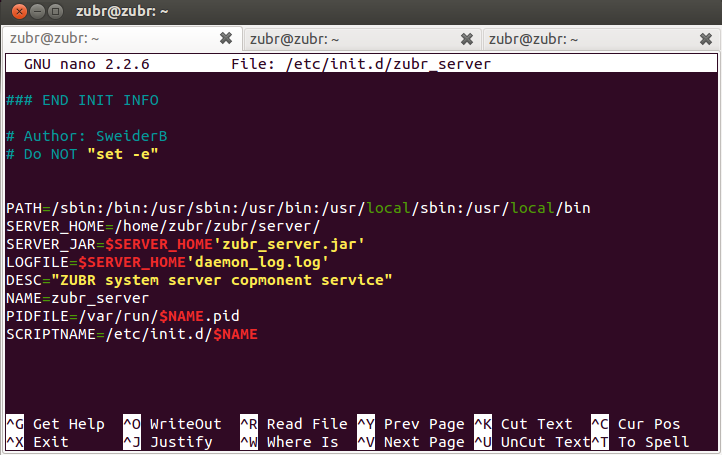
Для того, чтобы получить ссылку на java пакет воспользуемся командой

*which java*

Система отвечает, что нам доступна ссылка /bin/java:



Дальше создаем скрипт в папке /etc/init.d и настриваем необходимые переменные:

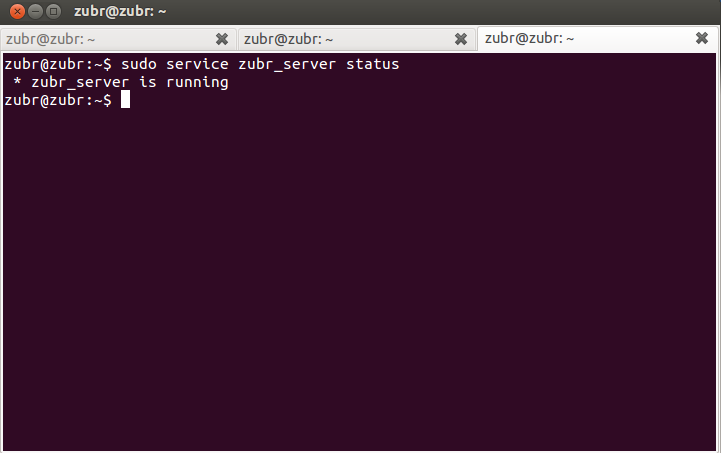


Дальше пишем основные функции – start, stop, status. Полный код скрипта можно посмотреть в приложении.

Регистрируем скрипт в системе командой

*update-rc.d zubr\_server defaults*

Перезагружаем систему и проверяем:



Система отвечает, что наш демон успешно запущен!

# **ЗАКЛЮЧЕНИЕ**

В процессе прохождения производственной практики на предприятии ООО «ЮГИС Групп» был приобретен опыт в сфере промышленной разработки программного обеспечения и применения технологий машинного обучения, приобретены знания об основных этапах жизненного цикла проекта.

Собрана информация о структуре предприятия, а также задачах, возлагаемых на каждый отдел и предприятие в целом. Была изучена система защиты информации на предприятии.

Изучены технологии разработки промышленного ПО на платформе Java, а так же адаптация этого ПО под работу в окружении ОС Linux.

# Приложение

Листинг кода скрипта автозапуска Java приложения zubr\_server:

#! /bin/sh

### BEGIN INIT INFO

# Provides: zubr\_server

# Required-Start: $remote\_fs $syslog

# Required-Stop: $remote\_fs $syslog

# Default-Start: 3 5

# Default-Stop: 0 1 2 4 6

# Short-Description: zubr\_server daemon

# Description: zubr core service

### END INIT INFO

# Author: SweiderB

# Do NOT "set -e"

PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin

SERVER\_HOME=/home/zubr/zubr/server/

SERVER\_JAR=$SERVER\_HOME'zubr\_server.jar'

LOGFILE=$SERVER\_HOME'daemon\_log.log'

DESC="ZUBR system server copmonent service"

NAME=zubr\_server

PIDFILE=/var/run/$NAME.pid

SCRIPTNAME=/etc/init.d/$NAME

# Load the VERBOSE setting and other rcS variables

. /lib/init/vars.sh

# Define LSB log\_\* functions.

# Depend on lsb-base (>= 3.2-14) to ensure that this file is present

# and status\_of\_proc is working.

. /lib/lsb/init-functions

echo "$(date) Defining functions." >> $LOGFILE

#

# Function that starts the daemon/service

#

do\_start()

{

echo "$(date) trying to start" >> $LOGFILE

#touch $PIDFILE

# Return

# 0 if daemon has been started

# 1 if daemon was already running

# 2 if daemon could not be started

if [ -e $PIDFILE ]; then

pid=`cat $PIDFILE`

process\_exists=`ps -e | grep -cE "\S?$pid"`

if [ $process\_exists -ne 0 ]; then

echo "$(date) Zubr\_server already running" >> $LOGFILE;

log\_failure\_msg "Already running!";

return 1;

else

echo "$(date) old pidfile with pid for not exist process found" >> $LOGFILE;

log\_warning\_msg "Found old pidfile"

rm $PIDFILE;

fi;

fi

touch $PIDFILE

CURRENT\_PWD=`pwd`

cd $SERVER\_HOME

/bin/java -jar $SERVER\_JAR > 'log/server\_output.log' &

echo $! > $PIDFILE

echo "$(date) started" >> $LOGFILE

log\_success\_msg "Started!"

cd $CURRENT\_PWD

return 0

}

#

# Function that stops the daemon/service

#

do\_stop()

{

echo "$(date) trying to stop server" >> $LOGFILE

# Return

# 0 if daemon has been stopped

# 1 if daemon was already stopped

# 2 if daemon could not be stopped

# other if a failure occurred

[ ! -e $PIDFILE ] && ( echo 'Cannot find zubr\_server pid file!' >> $LOGFILE; return 2; )

kill `cat $PIDFILE`

rm -f $PIDFILE

echo "$(date) returnning val is $RETVAL" >> $LOGFILE

echo "$RETVAL"

}

#

# Function that sends a SIGHUP to the daemon/service

#

do\_reload() {

#

# If the daemon can reload its configuration without

# restarting (for example, when it is sent a SIGHUP),

# then implement that here.

#

start-stop-daemon --stop --signal 1 --quiet --pidfile $PIDFILE --name $NAME

return 0

}

echo "$(date) current option is $1" >> $LOGFILE

VERBOSE="yes"

case "$1" in

start)

[ "$VERBOSE" != no ] && log\_daemon\_msg "Starting $DESC" "$NAME"

do\_start

case "$?" in

0|1) [ "$VERBOSE" != no ] && log\_end\_msg 0 ;;

2) [ "$VERBOSE" != no ] && log\_end\_msg 1 ;;

esac

;;

stop)

[ "$VERBOSE" != no ] && log\_daemon\_msg "Stopping $DESC" "$NAME"

do\_stop

case "$?" in

0|1) [ "$VERBOSE" != no ] && log\_end\_msg 0 ;;

2) [ "$VERBOSE" != no ] && log\_end\_msg 1 ;;

esac

;;

status)

status\_of\_proc "$DAEMON" "$NAME" && exit 0 || exit $?

;;

#reload|force-reload)

#

# If do\_reload() is not implemented then leave this commented out

# and leave 'force-reload' as an alias for 'restart'.

#

#log\_daemon\_msg "Reloading $DESC" "$NAME"

#do\_reload

#log\_end\_msg $?

#;;

restart|force-reload)

#

# If the "reload" option is implemented then remove the

# 'force-reload' alias

#

log\_daemon\_msg "Restarting $DESC" "$NAME"

do\_stop

case "$?" in

0|1)

do\_start

case "$?" in

0) log\_end\_msg 0 ;;

1) log\_end\_msg 1 ;; # Old process is still running

\*) log\_end\_msg 1 ;; # Failed to start

esac

;;

\*)

# Failed to stop

log\_end\_msg 1

;;

esac

;;

\*)

#echo "Usage: $SCRIPTNAME {start|stop|restart|reload|force-reload}" >&2

echo "Usage: $SCRIPTNAME {start|stop|status|restart|force-reload}" >&2

exit 3

;;

esac

exit 0