**Java EE** (Java Platform, Enterprise Edition), ранее известная как J2EE (Java 2 Platform, Enterprise Edition), является расширением платформы Java, предназначенным для разработки и выполнения распределенных, масштабируемых и надежных приложений предприятия.

Уровни:

**Уровень веб-клиента (Web Tier):**

Этот уровень предназначен для разработки веб-приложений.

Включает в себя веб-компоненты, такие как сервлеты и JSP (JavaServer Pages). Осуществляет взаимодействие с пользователем через веб-браузер.

**Уровень бизнес-логики (Business Tier):**

Этот уровень отвечает за бизнес-логику приложения.

Включает в себя компоненты EJB (Enterprise JavaBeans) для управления бизнес-процессами, транзакциями и другими аспектами корпоративных приложений.

**Уровень доступа к данным (Persistence Tier):**

Этот уровень отвечает за доступ к базам данных и управление данными.

Включает в себя технологии Java Persistence API (JPA) и Java Database Connectivity (JDBC).

**Уровень презентации (Presentation Tier):**

Этот уровень предназначен для отображения данных и взаимодействия с пользователем.

Включает в себя компоненты, такие как JSF (JavaServer Faces) для создания веб-интерфейсов и компоненты для обработки пользовательского ввода.

**Идемпотентность** — это свойство операции, которое означает, что повторное применение операции к объекту не изменяет его состояния или значение после первого применения.

web.xml — это дескриптор развёртывания и содержит конфигурационную информацию о веб-приложении.

Внутри файла web.xml определяются различные элементы, такие как сервлеты, URL-сопоставления, параметры контекста, фильтры и другие настройки, которые определяют поведение веб-приложения и его взаимодействие с клиентами.

build.xml — это файл конфигурации для управления сборкой проекта.

Внутри файла build.xml обычно определяются задачи (targets) и зависимости между ними.

Задачи могут включать в себя компиляцию кода, создание дистрибутивов, тестирование и другие этапы сборки проекта.

**Архитектуры**:

Монолитная архитектура (Monolithic Architecture): В этой архитектуре все компоненты приложения размещены в одном единственном монолитном приложении. Все части приложения взаимодействуют напрямую друг с другом. Это наиболее простая архитектура, но может стать сложной и трудно масштабируемой с ростом приложения.

Архитектура клиент-сервер (Client-Server Architecture): В этой архитектуре клиенты (например, веб-браузеры) обращаются к серверам (например, веб-серверам) для получения данных и выполнения бизнес-логики. Это позволяет легко масштабировать приложение, так как серверы могут быть горизонтально масштабируемыми.

Архитектура микросервисов (Microservices Architecture): Приложение разбивается на небольшие независимые микросервисы, каждый из которых выполняет конкретную функцию. Эти микросервисы взаимодействуют между собой через API. Это позволяет достичь высокой гибкости, масштабируемости и облегчает разработку и сопровождение.

Архитектура SOA (Service-Oriented Architecture): Это архитектурный стиль, в котором приложение строится из набора сервисов, которые могут предоставлять различные функции и ресурсы. Сервисы могут быть распределены по сети и могут быть использованы разными компонентами приложения.

Архитектура событийно-ориентированного программирования (Event-Driven Architecture): В этой архитектуре компоненты взаимодействуют через события. Когда что-то происходит в системе (событие), это может запускать обработчики событий в других компонентах. Это часто используется в системах реального времени и обработки данных.

Архитектура веб-служб (Web Services Architecture): В этой архитектуре приложение предоставляет свою функциональность через веб-службы, которые могут быть доступны через HTTP и другие стандарты. Это позволяет интегрировать разные системы и технологии.

Архитектура "событие-состояние" (Event Sourcing): В этой архитектуре изменения состояния приложения хранятся в виде событий, и текущее состояние может быть восстановлено путем воспроизведения событий. Это полезно для сложных систем, где требуется отслеживание истории изменений.

Компоненты Tomcat:

**Catalina** - контейнер сервлетов

**Jasper** - механизм JSP, роль шаблонизатора, генерирует нужный html код с использованием JSP компонентов.

**Coyote** - веб-сервер для обработки HTTP-запросов

**Servlet** — это Java-класс, который используется для создания динамических веб-приложений. Он обрабатывает HTTP-запросы и генерирует HTTP-ответы.

**От какого класса наследуется?**

javax.servlet.GenericServlet: Этот абстрактный класс является базовым классом для создания сервлетов. Он предоставляет базовую реализацию интерфейса Servlet и ServletConfig и может быть использован для создания сервлетов, работающих на низком уровне, например, для обработки протокола HTTP.

javax.servlet.http.HttpServlet: Этот абстрактный класс является подклассом GenericServlet и предназначен специально для создания HTTP-сервлетов. Он предоставляет методы для обработки HTTP-запросов (GET, POST, и так далее) и упрощает работу с данными HTTP-запросов и ответов.

**ServletConfig** — это интерфейс, который предоставляет информацию о настройках конкретного сервлета.

Этот интерфейс не наследуется от какого-либо класса и обычно предоставляется контейнером сервлетов (например, Apache Tomcat) при инициализации сервлета.

Он содержит методы для доступа к параметрам и контексту, специфичным для данного сервлета.

**Service** — это метод, который сервлеты должны переопределить для обработки HTTP-запросов.

Метод service принимает два параметра: объект ServletRequest, представляющий HTTP-запрос, и объект ServletResponse, представляющий HTTP-ответ.

Внутри метода service разработчики могут реализовывать логику обработки запросов и формирования ответов.

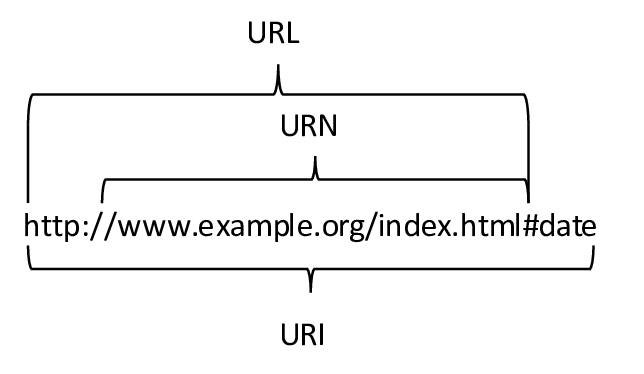
Этот метод обычно вызывается контейнером сервлетов.

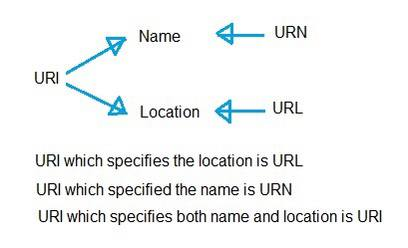
**Жизненный цикл Servlet**

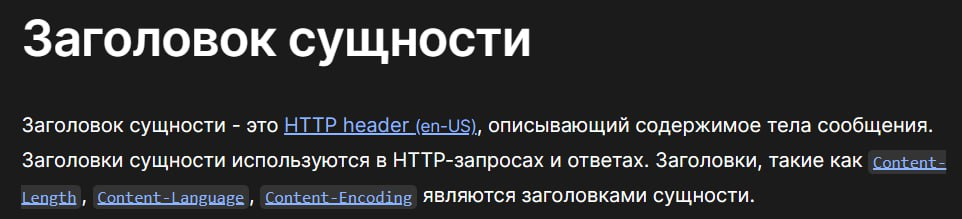
Инициализация (Initialization): Когда контейнер сервлетов первый раз загружает сервлет, вызывается метод init(ServletConfig config). Этот метод выполняется один раз при инициализации сервлета и используется для выполнения начальной настройки.

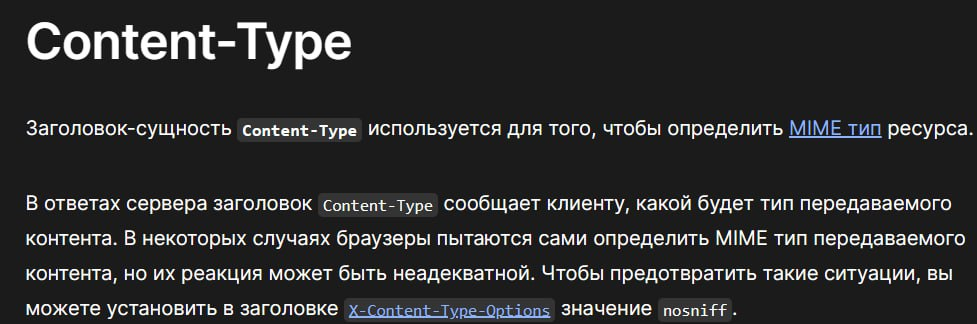
Обработка запросов (Request Handling): После инициализации, сервлет может обрабатывать множество HTTP-запросов. Каждый раз, когда поступает запрос от клиента, вызывается метод service(ServletRequest req, ServletResponse res). Этот метод обрабатывает запрос и отправляет ответ.

Завершение работы (Destroy): Когда сервлет больше не нужен (например, при завершении работы приложения или перезапуске контейнера сервлетов), вызывается метод destroy(). Этот метод позволяет сервлету освободить ресурсы и завершить работу.









Различия между версиями http:

1. HTTP/1.1:

- В HTTP/1.1 были внедрены улучшения, такие как поддержка постоянных соединений (keep-alive), мультиплексирование запросов, кэширование и виртуальные хосты.

- Эти улучшения значительно увеличили производительность и эффективность передачи данных.

2. HTTP/2:

- Он включает в себя многопоточность, мультиплексирование (возможность отправлять несколько запросов и ответов через одно соединение), сжатие заголовков и другие оптимизации.

- HTTP/2 обеспечивает значительно более быструю загрузку веб-страниц и ресурсов.

3. HTTP/3:

- Он основан на протоколе QUIC (Quick UDP Internet Connections) и использует UDP вместо TCP для более быстрой и надежной передачи данных.

- HTTP/3 включает в себя множество улучшений в производительности и безопасности, а также поддерживает потоковую передачу данных.

Основной целью обновления версий HTTP является улучшение производительности, безопасности и эффективности передачи данных через Интернет. HTTP/2 и HTTP/3 предоставляют значительные преимущества по сравнению с более старыми версиями протокола, и поэтому их применение становится все более распространенным для оптимизации работы веб-приложений и улучшения пользовательского опыта.

Java предоставляет несколько платформ и сред выполнения, помимо Java EE (Enterprise Edition). Вот краткое описание некоторых из них и их основных различий:

*1. Java SE (Standard Edition):*

- Java SE — это стандартная платформа для разработки и выполнения приложений на языке Java.

- Она включает в себя базовые библиотеки и API для общих задач, таких как работа с файлами, сетевое взаимодействие и многие другие.

- Java SE предназначена для разработки автономных приложений, настольных приложений и небольших серверных приложений.

*2. Java ME (Micro Edition):*

- Java ME — это платформа для разработки приложений, предназначенных для встроенных систем, мобильных устройств и других ограниченных окружений.

- Она оптимизирована для устройств с ограниченными ресурсами, такими как мобильные телефоны и встраиваемые системы.

- Java ME включает в себя подмножество библиотек и API Java SE.

*3. Java FX:*

- Java FX — это платформа для разработки интерактивных графических приложений и пользовательских интерфейсов (GUI).

- Она предоставляет инструменты и библиотеки для создания богатых визуальных приложений с поддержкой анимации и мультимедиа.

- Java FX часто используется для разработки приложений настольных систем и интерактивных веб-приложений.

*4. Java Card:*

- Java Card — это платформа для разработки приложений для устройств с ограниченными вычислительными ресурсами, таких как смарт-карты и SIM-карты.

- Эта платформа предоставляет минимальный набор библиотек и API для безопасной и эффективной работы на устройствах с ограниченными ресурсами.

*5. Java Embedded:*

- Java Embedded — это платформа, которая предоставляет Java для встроенных систем и IoT-устройств.

- Она позволяет разрабатывать Java-приложения для широкого спектра устройств, от датчиков до умных домов.