## Java-сервлеты. Особенности реализации, ключевые методы, преимущества и недостатки относительно CGI и FastCGI.

Сервлеты – это серверные сценарии, написанные на Java.
Сервлет — это класс, который расширяет функциональность класса HttpServlet и запускается внутри контейнера сервлетов.

В отличии от CGI, запросы обрабатываются в отдельных потоках, а не процессах, на веб-контейнере. (пример – Apache Tomcat)

Преимущества сервлетов:

- быстрее, чем CGI-сценарии
- хорошая масштабируемость
- надёжность и безопасность (поскольку реализованы на Java)
- платформенная независимость
- большое количество инструментов мониторинга и отладки

Недостатки сервлетов:

- слабое разделение уровня представления и бизнес-логики
- возможны конфликты во время параллельной обработки запросов

Для обработки запроса в HttpServlet определен ряд методов, которые мы можем переопределить в классе сервлета:

doGet: обрабатывает запросы GET (получение данных)
doPost: обрабатывает запросы POST (отправка данных)
doPut: обрабатывает запросы PUT (отправка данных для изменения)
doDelete: обрабатывает запросы DELETE (удаление данных)
doHead: обрабатывает запросы HEAD

Все методы в качестве параметра принимают два объекта: HttpServletRequest - хранит информацию о запросе и HttpServletResponse - управляет ответом на запрос.

## Контейнеры сервлетов. Жизненный цикл сервлета.

Жизненным циклом сервлета управляет веб-контейнер, он же контейнер сервлетов.
Для каждого сервлета движок сервлетов создает только одну копию. Вне зависимости от того, сколько запросов будет отправлено сервлету, все запросы будут обрабатываться толькой одной копией сервлета. Объект сервлета создается либо при запуске движка сервлетов, либо когда сервлет получает первый запрос. Затем для каждого запроса запускается поток, который обращается к объекту сервлета.

Init() – когда движок создаёт объект сервлета
service() – когда приходит запрос и вся работа здесь
destroy() – если объект сервлета долгое время не используется (к нему нет никаких запросов), или если происходит завершение работы движка, то движок вызывает этот метод и выгружает из памяти все созданные экземпляры сервлетов

## Диспетчеризация запросов в сервлетах. Фильтры сервлетов.

- Сервлеты могут делегировать обработку запросов другим ресурсам
- Диспетчеризация осуществляется с помощью реализаций интерфейса javax.servlet.RequestDispatcher
- Есть два способа получения RequestDispatcher – через ServletRequest (абсолютный или относительный URL) и ServletContext (только абсолютный URL)
- Есть два способа делегирования обработки запроса – forward и include.

- Фильтры позволяют осуществлять пред- и постобработку запросов до и после передачи их к ресурсу (примеры: допуск к странице только авторизованных пользователей – это предобработка, а запись в лог времени обработки запроса - постобработка)
- Реализуют интерфейс javax.servlet.Filter, ключевой метод которого – doFilter, который после выполнения передаёт управление следующему фильтру или целевому ресурсу, т.е. так можно реализовать последовательность фильтров для обработки одного запроса

## HTTP-сессии - назначение, взаимодействие сервлетов с сессией, способы передачи идентификатора сессии.

HTTP – stateless-протокол (протокол без сохранения состояния, т.е. каждая пара запрос-ответ не связана с предыдущими или последующими)

Сессия фактически «привязана» к конкретному приложению, соответственно у разных приложений – разные сессии.

Javax.servlet.HttpSession – интерфейс, позволяющий идентифицировать конкретного клиента (его браузер, но не устройство) при обработке множества запросов от него. Экземпляр этого интерфейса создаётся при первом обращении клиента к приложению и сохраняется на какое-то заданное разработчиком время после последнего обращения.

Идентификатор сессии хранится либо в cookie, либо добавляется в URL. Важно, что если удалить этот идентификатор, то сервер воспримет клиента как нового и создаст новую сессию.

В экземпляр HttpSession можно помещать общую для этой сессии информацию и работать с ней. Для этого используются методы getAttribute и setAttribute.

## Контекст сервлета - назначение, способы взаимодействия сервлетов с контекстом.

Контекст сервлета (да и вообще) – это API, которое предоставляет доступ к базовым функциям приложения и взаимодействия с контейнером: доступ к ресурсам, к файловой системе и т.д.

У всех сервлетов внутри приложения общий контекст, соответственно при помощи методов getAttribute и setAttribute можно помещать и получать общую для всех сервлетов информацию. Однако если приложение – распределённое, то на каждом экземпляре JVM контейнером создаётся свой контекст.

Доступ ко всем методам осуществляется через интерфейс javax.servlet.ServletContext.

## JavaServer Pages. Особенности, преимущества и недостатки по сравнению с сервлетами, область применения.

Страницы JSP – это текстовые файлы, содержащие статический HTML и позволяющие формировать динамическое содержание JSP-элементы.

При загрузке в веб-контейнер страницы JSP транслируются компилятором в сервлеты. Если комбинировать JavaServer Pages с сервлетами, то это позволит отделить бизнес-логику от уровня представления.

Преимущества:

- высокая производительность (поскольку транслируются в сервлеты)
- не зависят от используемой платформы (весь код пишется на Java)
- позволяют использовать JavaAPI (см. пояснение к предыдущему пункту)
- простые для понимания (структура схожа с обычным ~~языком программирования~~ HTML)

Недостатки:

- трудности в отладке, если приложение целиком основано на JSP
- возможны конфликты при параллельной обработке нескольких запросов

## Жизненный цикл JSP.

1) Трансляция JSP в код сервлета
2) Компиляция сервлета
3) Загрузка класса сервлета
4) Создание экземпляра сервлета (может быть при загрузке или при первом запросе)
5) Вызов метода jspInit() и ожидание запроса
6) После получения запроса вызов метода _jspService() и выполнение основного кода
7) Вызов метода jspDestroy()

## Структура JSP-страницы. Комментарии, директивы, объявления, скриптлеты и выражения.

Возможны два варианта синтаксиса – на базе HTML и XML. Для HTML используются теги <% %>

Есть 5 типов JSP-элементов:

- Комментарий `<%-- There is a comment --%>` (используется для пояснения кода)
- Директива `<%@ page import="java.util.*" %>` (предназначены для установки условий, которые применяются ко всей странице JSP)
- Объявление `<%! int sqr(int n){ return n*n; } %>` (позволяет определить метод, который впоследствии может быть вызван)
- Скриптлет `<% for (int i=1; i<5; i++) out.println("<li>"+sqr(i)+"</li>"); %>` (одна или более строчки кода на Java)
- Выражение `<%= sqr(8) %>` (вычисляется значение выражения в скобках)

## Правила записи Java-кода внутри JSP. Стандартные переменные, доступные в скриптлетах и выражениях.

В процессе трансляции контейнер добавляет в метод _jspService ряд объектов, которые можно использовать в скриптлетах и выражениях. Их называют предопределённые переменные.

Примеры: config, out, PageContext, session и другие.

По поводу правил записи Java-кода внутри JSP – важно соблюдать отступы (табуляцию) для всех видов элементов, т.е. как для HTML, так и для JSP. Внутри скриптлета правила записи кода не отличаются от записи кода на чистой Java.

## Bean-компоненты и их использование в JSP.

Класс Java Bean должен соответствовать ряду ограничений:

- иметь конструктор, который не принимает никаких параметров
- определять для всех свойств, которые используются в jsp, методы геттеры и сеттеры, названия которых должны соответствовать условностям (**is**Enabled для типа Boolean и **set**Name, **get**Name для остальных типов)
- реализовать интерфейс Serializable или Externalizable

Bean-компоненты хранят в себе бизнес-логику приложения.

## Стандартные теги JSP. Использование Expression Language (EL) в JSP.

Expression Language или сокращенно EL предоставляет компактный синтаксис для обращения к массивам, коллекциям, объектам и их свойствам внутри страницы jsp. Необходимые данные EL ищет через вставку ${…} во всех доступных контекстах, затем найденное значение (если оно всё-таки было найдено) конвертируется в строку и выводится на странице.

Для сложных объектов – массивов или списков – данные передаются аналогичным образом, например с указанием индекса элемента для массивов: ${users[0]}

## Параметры конфигурации JSP в дескрипторе развёртывания веб-приложения.

Параметры конфигурации JSP задаются в дескрипторе развёртывания (web.xml) внутри элемента jsp-config. Пример (Задаём количество элементов, которые, например, ArrayList будет выделять для начала. Как внутреннее хранилище элементов):

```
<jsp-config>
        <property name=”initial-capacity” value=”1024”>
</jsp-config>
```

## Шаблоны проектирования и архитектурные шаблоны. Использование в веб-приложениях.

Шаблон проектирования (паттерн) – повторимая архитектурная конструкция, которая представляет собой решение проблемы проектирования в рамках некоторого часто возникающего контекста. Т.е. паттерн описывает подход к решению типовой задачи, причём на одну и ту же задачу может подходить решение с использованием разных паттернов. Это нужно для упрощения жизни программистам – шаблоны позволяют избежать «типовых» ошибок, кратко описать подход к решению и упростить поддержку кода (его поведение более предсказуемо).

Архитектурные шаблоны имеют более высокий уровень, относительно шаблонов проектирования. Они описывают архитектуру всей системы или приложения, а значит имеют дело не с отдельными классами, а с целыми модулями, которые в свою очередь могут быть построены на разных шаблонах проектирования.

Веб-приложения имеют 3 уровня архитектуры:

- клиент
- бизнес-логика
- данные

## Архитектура веб-приложений. Шаблон MVC. Архитектурные модели Model 1 и Model 2 и их реализация на платформе Java EE.

Model 1:

- предназначена для проектирования приложений небольшого масштаба и сложности
- за обработку данных и представления отвечает один и тот же компонент: сервлет или JSP

Model 2:

- предназначена для проектирования достаточно сложных веб-приложений
- за обработку и представление данных отвечают различные компоненты

Шаблон MVC (Model-View-Controller или Модель-Представление-Контроллер) предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.