28) Интерфейс ServletContext. Интерфейс ServletConfig

.

ServletContext: Определяет набор методов которые использует сервлет для общения с контейнером сервлета.

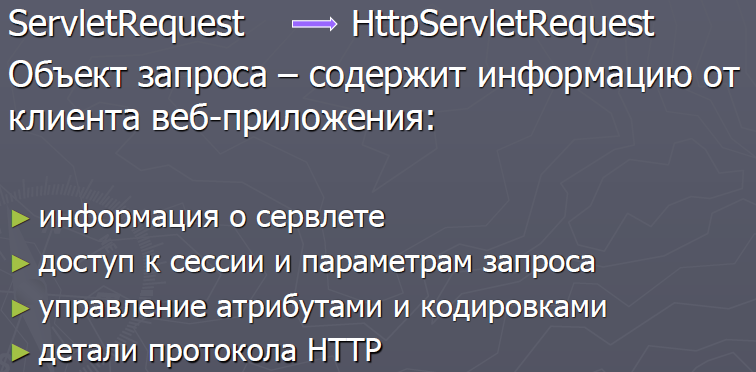
ServletConfig: объект конфигурации сервлета, используемый контейнером для передачи информации сервлету при инициализации

Интерфейс ServletConfig используется для передачи конфигурационной информации сервлету. Каждый сервлет имеет свой собственный объект ServletConfig, за создание экземпляра которого ответственен контейнер сервлетов. Для установки параметров конфигурации используются init параметры в web.xml (или аннотации WebInitParam). Для получения объекта ServletConfig данного сервлета используется метод getServletConfig().

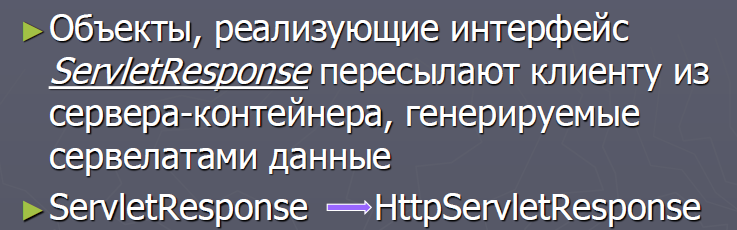
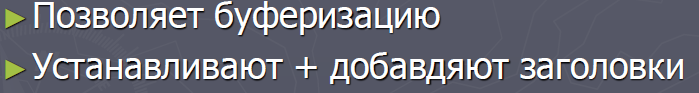
29) Интерфейс HttpServletRequest. Интерфейс HttpServletResponse

HttpServletRequest: Наследуется от ServletRequest. Представляет объект реквеста исходящего от клиента для HTTP сервлетов.

Контейнер сервлетов создает объект реквеста и передает в качестве аргумента в обслуживающие методы сервлета



HttpServletResponse: Наследуется от ServletResponse. Представляет объект ответа исходящего от сервера-контейнера к клиенту, содержащий данные сгенерируемые севрлетами.

Контейнер сервлетов создает объект респонза и передает в качестве аргумента в обслуживающие методы сервлета

30) Многопоточность в сервлете. Переадресация запросов и перенаправление откликов.

Для каждого клиента контейнер сервлетов создает независимый поток выполнения

Переадресация запросов актуальна только для текущего контекста, если нужно переадресоваться к странице в другом контексте, то используется перенаправление отклика.

Переадресация запроса: request.getRequestDispatcher(“Sevlet-Name-Here”).forward(request, response);

Перенаправление отклика: response.sendRedirect();

31) Сохранение данных приложения между запросами. Атрибуты и параметры.

Сохранение данных приложения между запросами возможно тремя способами:

1. присоединение объекта к запросу - HttpRequest.setAttribute(“Attribute Name”, attachedObject)
2. присоединение объекта к сессии – request.getSession().setAttribute(“Attribute Name”, attachedObject)
3. присоединение к контексту – getServletContext().setAttribute(“Attribute Name”, attachedObject)

Атрибут – объект, который может использоваться только на серверной стороне и существует в определенном пространстве (application, session, request, page). Может быть не только string.

Параметр – строковое значение, которое передается от клиента серверу и может быть получено из объекта реквеста

32) Сессия, события, файлы Cookie.

Сессия – соединение между клиентом и сервером, за которое клиент может отправить на сервер сколько угодно запросов. Время – 24 минуты

События – отслеживание изменений состояния объектов приложения(сеанс, контекст, запрос).

Для отслеживания событий используются интерфейсы.

Файлы Cookie – блоки текстовой информации, посылаемые сервером для сохранения в файлах Cookies (на стороне клиента). Могут быть запрещены клиентом. Возврат на сервер осуществляется путем присоединения к заголовку http при повторном заходе на ресурс.

Добавление: response.addCookie(CookieClassObj);

Получение: request.getCookies();

33) интерфейс Filter

Позволяет создать объект, который перехватывает запрос, может трансформировать заголовок и содержимое запроса или ответ сервера. Используется для предварительной обработки и модификации запроса до попадания к сервлету

Основные действия, которые может выполнить фильтр:

• перехват инициализации сервлета и определение содержания запроса, прежде чем сервлет будет инициализирован;

• блокировка дальнейшего прохождения пары request-response;

• изменение заголовка и данных запроса и ответа;

• взаимодействие с внешними ресурсами;

• построение цепочек фильтров;

• фильтрация более одного сервлета.

Основной метод – void doFilter(ServletRequest req, ServletResponse res, FilterChain chain), которому передаются объекты запроса, ответа и цепочки фильтров. Он вызывается каждый раз, когда запрос/ответ проходит через список фильтров FilterChain.

Кроме того, необходимо реализовать метод void init(FilterConfig config), который принимает параметры инициализации и настраивает конфигурационный объект фильтра FilterConfig. Метод destroy() вызывается при завершении работы фильтра, в тело которого помещаются команды освобождения используемых ресурсов.

Жизненный цикл фильтра начинается с однократного вызова метода init(), затем контейнер вызывает метод doFilter() столько раз, сколько запросов будет сделано непосредственно к данному фильтру. При отключении фильтра вызывается метод destroy().

34) Java Server Pages (JSP). Жизненный цикл. Неявные объекты в JSP.

JSP – серверный веб компонент JavaEE-приложения, предназначенный для обработки запроса, формирования и отправки ответа. Служит для создания страниц с динамическим содержанием.

Функции JSP:

1. отображение результатов работы приложения в виде текстовых документов типа html, xml, wml и др.
2. поддерживают JavaScript, HTML-теги
3. функциональные возможности

Содержит динамические и статические ресурсы.

Динамические ресурсы. Результаты их деятельности изменяются во время выполнения приложения. Обычно представлены в виде выражений Expression Language, библиотек тегов и тегов разработчика.

Статические ресурсы. Не изменяются в процессе работы (html, js, images, etc.)

Жизненный цикл: Когда пользователь переходит по ссылке, назовем ее index.jsp, он отправляет http запрос на сервер в виде "GET /index.jsp". Затем на основе этой страницы, сервер генерирует, компилирует и выполняет Java сервлет, который приходит ответом пользователю в виде отображения этой страницы. Такой жизненный цикл можно разделить на 6 шагов:

1. запрос от пользователя
2. чтение jsp страницы сервером
3. генерация java класса на основе этой страницы
4. компиляция в class файл
5. выполнение class файла
6. отправка ответа пользователю в виде html страницы

*Неявные объекты* (implicit objects) - это объекты, автоматически доступные как часть стандарта JSP без их специального объявления или импорта. Эти объекты, список которых представлен в таблице, можно использовать в коде JSP. (request, response, out, pageContext, session, application, config, page, exception)

35) Синтаксис страницы JSP. Директивы (directives);объявления (declarations); скриптлеты (scriptlets); выражения (expressions); комментарии (comments);

Страницы JSP имеют комбинированный синтаксис: объединение стандартного синтаксиса, соответствующего спецификации HTML, и синтаксиса JSP, определенного спецификацией Java Server Pages. Синтаксис JSP определяет правила записи страниц JSP, состоящих из стандартных тегов HTML и тегов JSP.

**Директивы** обеспечивают глобальную информацию, касающихся конкретных запросов, направляемых в JSP, и предоставляют информацию, необходимую на стадии трансляции.*Директивы* всегда помещаются в начале JSP-страницы до всех остальных тегов.

Синтаксис *директив* JSP выглядит следующим образом:

<%@ директива имяАтрибута="значение" %>

Директивы: page, taglib, include.

**Обьявления (Declarations)**предназначены для определения переменных и методов на языке скриптов, которые в дальнейшем используются на странице JSP. Синтаксис *declarations* имеет следующий вид :

<%! код Java %>

**Скриптлеты**включают различные фрагменты кода, написанного на языке скрипта, определенного в директиве *language*. Фрагменты кода должны соответствовать синтаксическим конструкциям языка *скриптлетов*, т.е., как правило, синтаксису языка Java. *Скриптлеты* имеют следующий синтаксис:

<% текст скриптлета %>

**Выражение** в странице JSP - это исполняемое выражение, написанное на языке скрипта, указанного в объявлении *language* (как правило Java). Результат *выражения*JSP, имеющий обязательный тип *String*, направляется в стандартный поток вывода *out*с помощью текущего объекта *JspWriter*. Если результат *выражения* не может быть приведен к типу *String*, возникает либо ошибка трансляции, если проблема была выявлена на этапе трансляции, либо возбуждается исключение *ClassCastException*, если несоответствие было выявлено в процессе выполнения запроса. *Выражение* имеет следующий синтаксис:

<%= текст выражения %>

**Комментарии**. В страница JSP можно использовать два типа комментариев:

1. Выводимый комментарий (output comment).
2. Закомментированный блок (commented block).

*Выводимый комментарий* - это такой стиль комментирования, при котором комментарий выводится в выходной поток и отображается в браузере. Синтаксис комментариев данного типа следующий:

<!-- Текст комментария -->

*Закомментированный блок* - это стиль комментирования, при котором тело закомментированного блока полностью игнорируется, не выводится в выходной поток и, следовательно, не отображается в браузере. Этот стиль обычно используется для поясняющих записей, касающихся фрагментов программного кода страницы JSP.Закомментированный блок имеет следующий синтаксис: <%-- Текст комментария --%>

36) Стандартные Action - теги JSP. jsp:useBean, jsp:setProperty, jsp:getProperty, jsp:include, jsp:forward, jsp:param. Expression Language (EL).

**jsp:useBean** – позволяет использовать экземпляр компонента Java Bean. Если экземпляр с указанным идентификатором не существует, то он будет создан с областью видимости **page** (страница), **request** (запрос), **session** (сессия) или **application**(приложение). Объявляется, как правило, с атрибутами **id**(имя объекта), **scope** (область видимости), **class**(полное имя класса), **type** (по умолчанию **class**).

**<jsp:useBean id=”ob”** **scope=”session” class=”test.MyBean” />**

**jsp:setProperty**– позволяет устанавливать значения полей указанного в атрибуте **name** объекта. Если установить значение **property** в «**\***», то значения свойств компонента JavaBean будут установлены таким образом, что будет определено соответствие между именами параметров и именами методов-установщиков (setter-ов) компонента:

**<jsp:setProperty name=”ob”** **property=”field” value=”привет” />**

**jsp:getProperty**– получает значения поля указанного объекта, преобразует его в строку и отправляет в неявный объект **out**:

**<jsp:getProperty name=”ob” property=”field” />**

**jsp:include**– позволяет включать файлы в генерируемую страницу при запросе страницы:

**<jsp:include page=”**относительный URL**”** **flush=”true”/>**

**jsp:forward**– позволяет передать запрос другой странице:

**<jsp:forward page=”**относительный URL**”/>**

**jsp:param**– добавляет параметры в объект запроса, например в элементах **forward**, **include**, **plugin**.

Expression Language (EL) – упрощение доступа к данным(атрибутам, параметрам, хранящимся в различных областях видимости) и вычисления простых выражений. Поддерживается «.» и «[]».

${ attribure\_name}

Операторы: ==, !=, <, >, <=, >=, +, -, \*, /, %, &&, ||, !, empty

37) Библиотека тегов JSTL. Стандартные теги: core, formatting, sql, xml, functions.

Стандартная библиотека тегов JSP (JavaServer Pages Standard Tag Library, **JSTL**) является расширением её спецификации, используемая для условной обработки, создания циклов, интернационализации страницы, разбора XML данных.

**JSTL** является альтернативой скриплетам, встроенным в JSP, то есть прямым вставкам Java кода. **JSTL** представляет собой набор тегов в стиле HTML, позволяющих обращаться к объектам Java и выполнять многие из конструкций языка Java.

// Основные теги создания циклов, определения условий, вывода информации на страницу и т.д.

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

// Теги для работы с XML-документами

<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>

// Теги для работы с базами данных

<%@ taglib prefix="s" uri="http://java.sun.com/jsp/jstl/sql" %>

// Теги для форматирования и интернационализации информации (i10n и i18n)

<%@ taglib prefix="f" uri="http://java.sun.com/jsp/jstl/fmt" %>

Тэг **<c:out>** используется для вывода данных на экран. Следующий код отобразит выражение, представленное в атрибуте value.

Тег **c:set** используется для определения переменной подобно <[jsp:setProperty](http://java-online.ru/jsp-javabean.xhtml)>.

Для удаления переменной следует использовать тэг **<c:remove>**.

Тег **<c:if>** серьезно отличается от того, что мы используем в других языках программирования. Он не позволяет определить ветку else или elseif. Можно только проверить некоторое условие.

Тег **<c:choose>** в Java является аналогом оператора switch.

Тег **<c:forEach>** позволяет задать цикл.

Для вложения одной страницы в другую можно использовать тег **<c:import>**.

Для обработки исключений в теле JSP следует использовать тэг **<c:catch>**.

|  |  |
| --- | --- |
| <fmt:formatNumber> | To render numerical value with specific precision or format. |
| <fmt:parseNumber> | Parses the string representation of a number, currency, or percentage. |
| <fmt:formatDate> | Formats a date and/or time using the supplied styles and pattern |
| <fmt:parseDate> | Parses the string representation of a date and/or time |
| <fmt:bundle> | Loads a resource bundle to be used by its tag body. |
| <fmt:setLocale> | Stores the given locale in the locale configuration variable. |
| <fmt:setBundle> | Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable. |
| <fmt:timeZone> | Specifies the time zone for any time formatting or parsing actions nested in its body. |
| <fmt:setTimeZone> | Stores the given time zone in the time zone configuration variable |
| <fmt:message> | To display an internationalized message. |
| <fmt:requestEncoding> | Sets the request character encoding |

Библиотеки SQL: <sql:query>, <sql:update>, <sql:setDataSource>, <sql:param>, <sql:dateParam>

Библиотеки XML:  
        Общие XML операции: <x:parse>, <x:out>, <x:set>  
        XML-контроль: <x:if>, <x:choose>, <x:when>, <x:otherwise>, <x:forEach>  
        Трансформации: <x:transform>, <x:param>

38) Библиотека JSTL. Пользовательские теги.

Разработка пользовательского тега состоит из нескольких шагов:  
1.    Написать класс обработчика тега.  
2.    Создать? описатель библиотеки тегов (TLD).  
3.    Открыть доступ к файлу TLD и классам обработчика.  
4.    Сослаться на библиотеку тегов.  
5.    Использовать тег на странице JSP.

Для создания тега без атрибутов или тела необходимо переопределить метод doStartTag(), определяющий код, который вызывается во время запроса, если обнаруживается начальный элемент тега

Тег может содержать параметры и передавать их значения для обработки в соответствующий ему класс. Для этого при описании тега в файле \*.tld используются атрибуты, которые должны объявляться внутри элемента tag с помощью элемента attribute.

//хз что писать//

39) Maven. Назначение. Состав объектной модели проекта [pom.xml](https://www.dropbox.com/referrer_cleansing_redirect?hmac=Jj5PbrwR7bkyVI9EoysZyB4V2PKabkkDECoZgsa70%2FE%3D&url=http%3A%2F%2Fpom.xml" \t "_blank)

**Apache Maven** — [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) для [автоматизации сборки](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%81%D0%B1%D0%BE%D1%80%D0%BA%D0%B8) проектов на основе описания их структуры в файлах на языке [POM](https://en.wikipedia.org/wiki/Project_Object_Model), являющемся подмножеством [XML](https://ru.wikipedia.org/wiki/XML)

Вся структура проекта описывается в файле pom.xml (POM – Project Object Model), который должен находиться в корневой папке проекта. Ключевым понятием Maven является **артефакт** — это, по сути, любая библиотека, хранящаяся в репозитории. Это может быть какая-то зависимость или плагин.   
  
**Зависимости** — это те библиотеки, которые непосредственно используются в вашем проекте для компиляции кода или его тестирования.  
  
**Плагины** же используются самим Maven'ом при сборке проекта или для каких-то других целей (деплоймент, создание файлов проекта для Eclipse и др.).  
  
В самом начале работы с Maven, пользователь непременно столкнется с таким понятием как архетип. **Архетип** — это некая стандартная компоновка файлов и каталогов в проектах различного рода (веб, swing-проекты и прочие). Другими словами, Maven знает, как обычно строятся проекты и в соответствии с архетипом создает структуру каталогов.

40) Основные фазы сборки проекта maven. Жизненный цикл сборки. Понятие профайла сборки. Управление зависимостями. Maven plugin

Жизненный цикл у мавена довольно ожидаемый:

validate — проверяет корректность метаинформации о проекте

* compile — компилирует исходники
* test — прогоняет тесты классов из предыдущего шага
* package — упаковывает скомпилированые классы в удобноперемещаемый формат (jar или war, к примеру)
* integration-test — отправляет упаковынные классы в среду интеграционного тестирования и прогоняет тесты
* verify — проверяет корректность пакета и удовлетворение требованиям качества
* install — загоняет пакет в локальный репозиторий, откуда он (пекат) будет доступен для использования как зависимость в других проектах
* deploy — отправляет пакет на удаленный production сервер, откуда другие разработчики его могут получить и использовать

Профиль сборки – это множество настроек, которые могут быть использованы для установки или перезаписи стандартных значений сборки Maven.