**Java Servlet API** — стандартизированный API, предназначенный для реализации на сервере и работе с клиентом по схеме запрос-ответ.

Сервлет представляет специальный тип классов Java, который выполняется на веб-сервере (контейнере сервлетов) и который обрабатывает запросы и возвращает результат обработки.

Абстрактный класс **GenericServlet** – независимая от используемого протокола реализация интерфейса Servlet, а абстрактный класс **HttpServlet** расширяет GenericServlet для протокола HTTP.

Чтобы создать сервлет мы наследуемся от класса **HttpServlet.**

Аннотация **WebServlet,** указывает какой урл будет обрабатывать этот сервлет.



**Методы сервлетов**:

* **getServletConfig() -** возвращает ссылку на объект, который предоставляет доступ к информации о конфигурации сервлета; Конфиг позволяет получить параметры инициализации сервлета, имя сервлета и его контекст.
* **getServletInfo()** возвращает строку, содержащую информацию о сервлете, например: автор и версия сервлета;

**Преимущества сервлетов:**

* Хорошая производительность обработки запросов, и более эффективное использование памяти за счет использования преимущество многопоточности (на каждый запрос создается новая нить, вместо создания нового объекта).
* Сервлеты повышают надежность программы, так как контейнер сервлетов самостоятельно заботится о жизненном цикле.
* Контейнер сервлетов заботится о создании слушателя сокета на сервере, разбирает запрос и его параметры, преобразовывая его в объект и генерирует ответ. Разработчик может сосредоточится на бизнес логике приложения.

Так как сервлеты используют многопоточность, то стоит задуматься о **Потокобезопасности**. Доступ к глобальным разделяемым ресурсам (поля объекта или класса) следует синхронизировать. А по возможности лучше вообще от них отказаться.

Для обработки Http-запросов 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
* **doOptions()**: обрабатывает запросы OPTIONS
* **doTrace():** обрабатывает TRACE/

Все эти методы принимают 2 параметра:

* **HttpServletRequest** – инкапсулирует всю информацию о запросе.
* **HttpServletResponce** – позволяет управлять ответом.

**Методы HttpServletRequest**:

* String **getParameter(String param)** – получить значение параметра по имени. Если указанного параметра в запросе нет, возвращается null. Используется для получения данных из форм, либо адресной строки.  
  
* String[] **getParameterValues(String param)** – возвращает массив значений определенного параметра. Если указанного параметра нет, возвращает null.
* String[] **getParameterNames()** – получить имена всех параметров, отправленных в запросе.
* Cookie[] **getCookies()** – получить куки, которые приходят от клиента.
* **HttpSession getSession() –** позволяет получить сессию.
* **getRemoteAddr()** – получить ip-адрес клиента.
* **getAttributeNames()** – перечисление, которое содержит имена всех доступных атрибутов данного запроса.
* **getReader() –** получить текстовый поток, для чтения тела запроса.

**Методы HttpServletResponse**

* **sendRedirect(adress) –** метод для переадресации. Адрес может быть локальным, внутренним, а может быть и внешним.
* **addCookie(Cookie c) –** добавить куки в ответ клиенту. Объект куки создается из имени и значения - Cookie(String name, String value)
* **getOutputStream, getWriter –** возвращают потоки вывода, через которые можно записать что-то в ответ клиенту. OutputStream – бинарный, Writer – текстовый.
* **setContentType(String type) –** указать MIME-тип ответа браузеру, чтобы он мог определить как отображать данные.
* **encodeRedirectURL(String url) –** кодирует указанный урл для использования в методе redirect
* **encodeURL(String url) –** кодирует урл, включая в него идентификатор сессии.
* **addHeader(String name, String value) –** добавляет заголовок с указанным именем и значением.
* **sendError(int sc) –** отправляет клиенту ошибку с указанным кодом и очищает буфер.
* **sendError(int sc, String msg) -** Отправляет клиенту ошибку с указанным кодом и сообщение.
* **setStatus(int sc) –** установить статус ответа.

**SERVLETCONTEXT**

Метод **getServletContext()** позволяет получить контекст сервлета. Позволяет получать информациб о среде выполнения, а также использовать ресурсы совместно с другими объектами приложения.

**Методы класса ServletContext**:

**void setAttribute(String name, Object value) —** добавляет атрибут

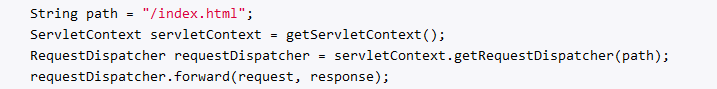
**getAttribute(String name) —** получает значение атрибута по имени;

**getAttributeNames() —** получает список имен атрибутов;

**removeAttribute(String name) —** удаляет атрибут из контекста

**getContextPath() –** позволяет получить путь к текущему приложению.

Метод **getRequestDispatcher** позволяет получить объект RequestDispetcher с помощью которого можно выполнить перенаправление запроса.



И если мы обратимся к сервлету, то фактически мы получим содержимое страницы index.html, который будет перенаправлен запрос.

Подобным образом мы можем выполнять перенаправление на страницы jsp и другие сервлеты.

Forward не меняет урл в браузере и выполняется на сервере, в отличие от redirect, который отправляет браузеру ответ с кодом 302 и браузер делает новый запрос на новый адрес.

Помимо перенаправления, можно включить контент ресурса в ответ с помощью **include**.

**getInitParameter(name)** – получить параметр инициализации (эти параметры конфигурируются в web.xml в теге <context-param>).

**getRealPath(request.getServletPath()) –** получить реальный путь к расположению сервлета на сервере.

**getServerInfo()** – получить информацию о сервере.

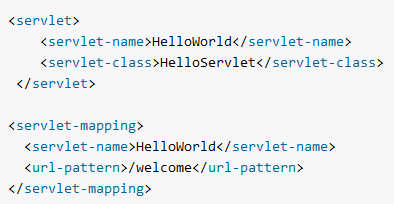
**МАППИНГ**

Чтобы указать, какой url обрабатывает сервлет есть несколько способов.

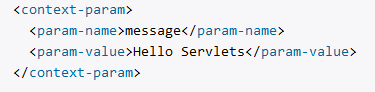
1. С помощью аннотации **@WebServlet("/hello")**.
2. С помощью файла **web.xml (дескриптор развертывания).**

Файл web.xml хранит информацию о конфигурации приложения, однако он не является обязательным. При запуске Tomcat считывает его содержимое, и использует считанную конфигурацию.

Корневой узел <**web-app>.** Внутри мы можем указывать какие есть сервлеты, и мапинги к ним.

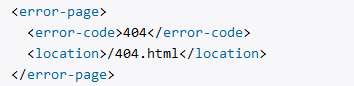


**<context-param> -** создает параметр, который будет доступен всем сервлетам в приложении.

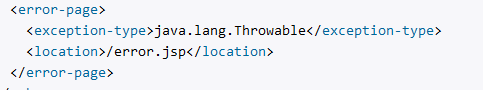


<**init-param> -** параметр, который доступен только одному сервлету. Пишется внутри тега <servlet>.

<**error-page> -** страница, которая будет отправляться пользователю при отправке статусных кодов ошибки. Например для вывода сообщения о том что ресурс не найден:



Либо можно обрабатывать java-исключения



**ЖИЗНЕННЫЙ ЦИКЛ СЕРВЛЕТА**

Для каждого сервлета контейнер сервлетов создает только одну копию. Вне зависимости от того, сколько запросов будет отправлено сервлету, все они будут обрабатываться только одной копией. Объект сервлета создается либо при запуске контейнера, либо когда сервлет получает первый запрос. Затем для каждого запроса запускается отдельный поток, который обращается к объекту сервлета.

1. Когда движок создает объект сервлета, у сервлета вызывается метод **init()**. Этот метод внедряет ServletConfig в объект сервлета. Мы можем переопределить этот метод, чтобы определить в нем некоторую логику инициализации.
2. Когда приходит запрос, движок вызывает метод **service()** сервлета в отдельном потоке. Этот метод исходя из типа запроса решает, какому методу обрабатывать запрос.
3. Если объект сервлета долгое время не используется, или если происходит завершение работы контейнера с сервлетами вызывается метод **destroy()**, который используется при выгрузке экземпляра сервлета из памяти. При необходимости мы также можем его переопределить, например, определить в нем логику логгирования или что иное. Следует помнить, что при аварийном завершении данный метод естественно вызван не будет.

Не рекомендуется создавать глобальные переменные в классе сервлета, так как они не будут потокобезопасными.

Если необходимо загрузить сервлет прямо **на старте приложения** (например если загрузка сервлета происходит длительное время) следует использовать элемент **<load-on-startup>** в дескрипторе или аннотацию **@loadOnStartup** в коде сервлета, что будет указывать на необходимость загрузки сервлета при запуске.

Чем число меньше, тем раньше в очереди на загрузку оно окажется.

Если число отрицательное, то сервлет будет загружен при запросе клиента.



**HTTP-SESSION**

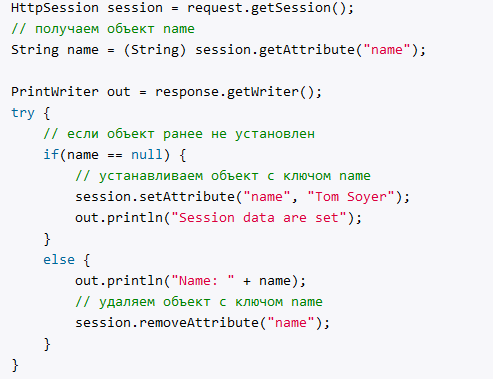
**Сессия** - это сеанс связи между клиентом и сервером, устанавливаемый на определенное время.

Сессия позволяет сохранять некоторую информацию на время сеанса, между просмотрами отдельных страниц. Когда клиент обращается к сервлету или странице JSP, то движок сервлетов проверяет, определен ли в запросе параметр ID сессии. Если такой параметр неопределен (например, клиент первый раз обращается к приложению), тогда движок сервлетов создает уникальное значение ID и связанный с ним объект сессии. Объект сессии сохраняется на сервере, а ID оправляется в ответе клиенту и по умолчанию сохраняется на клиенте в куках. Затем когда приходит новый запрос от того же клиента, то движок сервлетов опять же может получить ID и сопоставить его с объектом сессии на веб-сервере.

**Методы HttpSession**

* **setAttribute(String name, Object o)**: сохраняет в сессии под ключом name
* **getAttribute(String name)**: возвращает из сессии объект с ключом name. Если ключа name в сессии неопределено, то возвращается null
* **removeAttribute(String name)**: удаляет из сессии объект с ключом name
* **getAttributeNames()**: возвращает объект java.util.Enumeration, который содержит все ключи имеющих в сессии объектов
* **getId():** возвращает идентификатор сессии в виде строки
* **isNew():** возвращает true, если для клиента еще не установлена сессия (клиент сделал первый запрос или на клиенте отключены куки)
* **setMaxInactiveInterval(int seconds)**: устанавливает интервал неактивности в секундах. И если в течение этого интервала клиент был неактивен, то данные сессии данные удаляются. По умолчанию максимальный интервал неактивности 1800 секунд. Значение -1 указывает, что сессия удаляется только тогда, когда пользователь закрыл вкладку в браузере.
* **invalidate()**: удаляет из сессии все объекты, делает ее недействительной.

Удобнее всего для проверки валидности сессии использовать фильтр, и потом уже давать доступ к нужному сервлету.



**ФИЛЬТРЫ**

**Сервлетный фильтр** – Java-код, позволяющий преобразовывать содержимое http-запросов и ответов. Он занимается предварительной обработкой запроса, перед тем как запрос попадет в сервлет, либо обработкой ответа сервлета.

Фильтры могут:

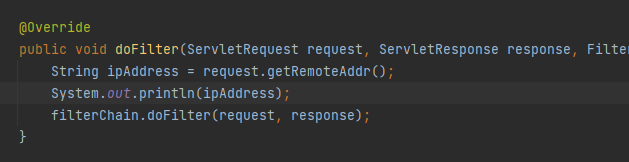
* Перехватывать инициацию сервлета
* Определить содержимое запроса прежде, чем сервлет будет инициирован.
* Модифицировать заголовки и данные запроса и ответа.

Фильтр может работать с одним сервлетом либо с группой.

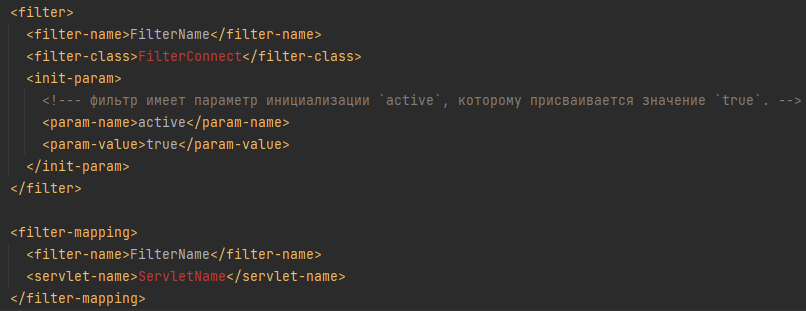
**javax.servlet.Filter** основной интерфейс для работы с фильтрами, в котором объявлены три метода:

* **init() –** инициализация фильтра.
* **doFilter()** – каждый фильтр получает текущий запрос (request) и ответ (response), а также FilterChain, содержащий список фильтров, предназначенных для обработки.

В этом методе фильтр выполняет свою работу, затем вызывает метод chain.doFilter(), чтобы передать управление следующему фильтру. После возвращения этого вызова фильтр может выполнить дополнительную работу над полученным ответом.



Объявление фильтра и подключение его к сервлету через web.xml:



Также можно использовать аннотацию **@WebFilter:**



Контейнер сервлетов строит цепочку фильтров в той последовательности, в которой они объявлены в файле.

**LISTENER**

**Listner –** работает как триггер, выполняя определенные действия при наступлении какого-либо события в жизненном цикле сервлета.

Например, можно отслеживать время входа и выхода из системы пользователей, действия, выполняемые пользователем. Осуществлять инициализацию ресурсов в момент запуска системы, создания пользовательской сессии/

Слушатели разделены по области видимости (scope):

**Request:**

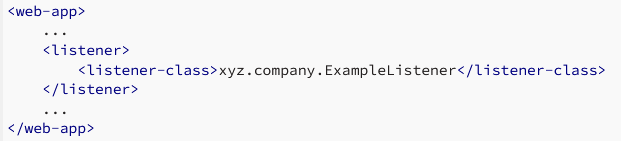
* **ServletRequestListener** используется для того, чтобы поймать момент создания и уничтожения запроса;
* **ServletRequestAttributeListener** используется для прослушивания событий происходящих с атрибутами запроса.

**Context:**

* **ServletContextListener** позволяет поймать момент, когда контекст инициализируется либо уничтожается;
* **ServletContextAttributeListener** используется для прослушивания событий происходящих с атрибутами в контексте.

**Session:**

* **HttpSessionListene**r позволяет поймать момент создания и уничтожения сессии;
* **HttpSessionAttributeListener** используется при прослушивании событий происходящих с атрибутами в сессии;
* **HttpSessionActivationListener** используется в случае, если происходит миграция сессии между различными JVM в распределённых приложениях;
* **HttpSessionBindingListener** так же используется для прослушивания событий происходящих с атрибутами в сессии. Разница между HttpSessionAttributeListener и HttpSessionBindingListener слушателями: первый декларируется в web.xml; экземпляр класса создается контейнером автоматически в единственном числе и применяется ко всем сессиям; второй: экземпляр класса должен быть создан и закреплён за определённой сессией «вручную», количество экземпляров также регулируется самостоятельно.
* **HttpSessionActivationListener** — обрабатывает события, связанные с активацией/деактивацией HTTP-сессии;

**Подключение слушателей:**

*HttpSessionBindingListener* подключается в качестве атрибута непосредственно в сессию, т.е., чтобы его подключить необходимо:

* создать экземпляр класса реализующего этот интерфейс;
* положить созданный экземпляр в сессию при помощи *setAttribute(String, Object).*

Следует **использовать фильтры**, если необходимо **обрабатывать входящие или исходящие данные** (например: для аутентификации, преобразования формата, компрессии, шифрования и т.д.), в случае, когда необходимо **реагировать на события** - лучше **применять слушателей**.

**JSP**

**Java Server Pages** технология, которая позволяет создавать динамические веб-страницы. JSP представляют собой html-код с вкраплениями кода Java.

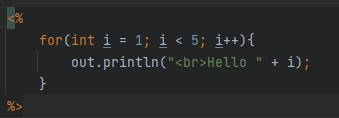
Контейнер сервлетов преобразует каждую jsp в сервлет c помощью метода **service**() и управляет ей как обычным сервлетом.

Когда происходит запрос к определенной странице JSP, то сервер обрабатывает ее, генерирует из нее html код и отправляет клиенту.

**JSP Expression** представляет выражение, заключенное между тегами **<%=** и **%>**. При обращении к JSP вычисляется значение этого выражения, которое будет выводится вместо этих тегов.



**JSP Scriplet** представляет одну или несколько строк на языке Java. Скриплет заключается внутри **<% … %>**:

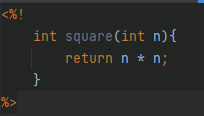


Скриплет можно начать в одном месте, затем вставить какой-нибудь html и затем закрыть.



**JSP Declaration** позволяют определить метод, который мы затем можем вызывать в скриплетах или в JSP-выражениях. Определение метода помещается между тегами **<%! и %>.**

При преобразовании jsp в сервлет Declaration преобразуется в метод класса.



**Директивы** предназначены для установки условий, которые применяются ко всей странице JSP. Наиболее используемая директива - это директива **page**. Например, с помощью атрибута **import** этой директивы мы можем импортировать пакеты или отдельные классы на страницу jsp.



Другой часто используемый атрибут - **pageEncoding**, который задает кодировку jsp.



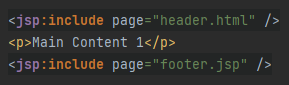
Директива **include** позволяет подключить файл к jsp. При этом подключение происходит при трансляции jsp в сервлет.



**Комментарии** JSP добавляются с помощью тега <%-- Текст\_комментария --%>. При этом внутри скриплета мы можем использовать стандартные для языка Java комментарии.



**<jsp:include page=”pagename” /> -** специальный тег, позволяющий включать в jsp страницу другие html или jsp страницы. В отличии от директивы include подключается при запросе страницы.



**<jsp:forward page="относительный URL"/>** - передать запрос другой странице.

Существуют неявные объекты, которые создаются контейнером и могут быть использованы на jsp странице:

* <% **out**.print(new Date()); %>
* %=**request**.getHeader("User-Agent") %>
* <%**response**.addCookie(new Cookie("Test","Value")); %>
* <%=**config**.getInitParameter("User") %>
* <%=**application**.getInitParameter("User") %>
* <%=**session**.getId() %>
* <% **pageContext**.setAttribute("Test", "Test Value"); %>
* <%=**page**.getClass().getName() %>
* <%=**exception** %> - доступен только на странице ошибок.

**ПЕРЕДАЧА ДАННЫХ ИЗ СЕРВЛЕТА В JSP**

Способы передачи данных из сервлета в jsp заключаются в использовании определенного контекста или **scope.**

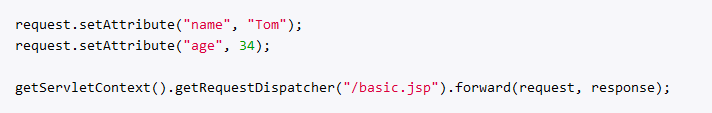
**Виды Scope:**

* **request** (контекст запроса): данные сохраняются в HttpServletRequest
* **session** (контекст сессии): данные сохраняются в HttpSession
* **application** (контекст приложения): данные сохраняются в ServletContext
* **page** (область действия страницы) - объект будет доступен только на той странице, где он определен. На включаемых (jsp:include) и переадресуемых (jsp:forward) страницах данный объект уже не будет доступен.

Для указания требуемой области действия при определении объекта на JSP странице используется **атрибут scope** тега jsp:useBean



Наиболее распространенный способ передачи данных с помощью метода атрибутов запроса HttpServletRequest. Записываем туда аттрибуты, и можем получить их в сервлете через **${name}**.





**EL, JSTL**

**Expression Language –** Специальный синтаксис, позволяющий получать данные в jsp с помощью синтаксиса **${name}**.

Некоторые версии могут игнорировать этот синтаксис, чтобы это исправить можно добавить директиву:



EL пытается найти значения для этих данных во всех доступных контекстах в следующем порядке:

1. Контекст страницы (данные сохраняются в PageContext)
2. Контекст запроса
3. Контекст сессии
4. Контекст приложения

По умолчанию EL предоставляет ряд встроенных объектов, которые позволяют использовать различные аспекты запроса:

* **param**: объект, который хранит все переданные странице параметры
* **paramValues**: хранит массив значений для определенного параметра (если для параметра передается сразу несколько значений)
* **header**: хранит все заголовки запроса - ${header["User-Agent"]}
* **headerValues**: предоставляет массив значений для определенного заголовка запроса
* **cookie**: предоставляет доступ к отправленным в запросе кукам
* **initParam**: возвращает значение для определенного параметра из элемента context-param из файла web.xml
* **pageContext**: предоставляет доступ к объекту PageContext, который представляет контекст текущей страницы jsp
* **pageScope** Контекст страницы. Переменные доступны только для текущей страницы.
* **requestScope** Контекст запроса. Переменные доступны на всех страницах, а также сервлетам, обслуживающим текущий запрос пользователя.
* **sessionScope** Контекст сессии. Переменные доступны в течение всей сессии пользователя, т.е. пока не будет закрыт браузер или не закончится предельное время бездействия.

**JSTL (JSP Standard Tag Library)** предоставляет теги для базовых задач JSP (цикл, условные выражения).

Содержит в себе ряд библиотек:

* **Core**: содержит основные теги для наиболее распространенных задач: итерации, обработки исключений, URL, forward, redirect response и т.д.  
  Использует префикс "c"
* **Formatting**: предоставляет теги для форматирования чисел, дат, времени.  
  Использует префикс "fmt"
* **SQL**: предоставляет теги для работы с sql-запросами и источниками данных.  
  Использует префикс "sql"
* **XML**: предоставляет теги для работы с xml.  
  Использует префикс "x"
* **Functions**: предоставляет функции для работы со строками.  
  Использует префикс "fn"

Для подключения функционала этих библиотек на страницу jsp применяется директива **taglib**.



**Плюсы:**

* Защита от внедрения кода (EL не позволяет нам экранировать теги)  
    
  тег **out** позволяет передавать значение по умолчанию  
  

**<c:set>** - определение переменной

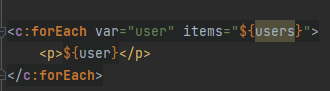


можно устанавливать свойства компонентов JavaBean, добавлять значения к спискам и массивам. Для этого используется параметр target.

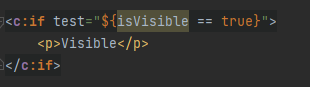


**<c:remove> -** удаление переменной.

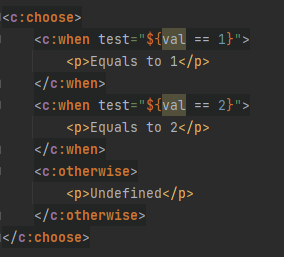
Цикл:



Оператор **if**:



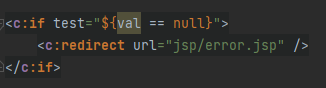
Тег **choose** (аналог switch case):



Тег **url** позволяет создать адрес относительно корня приложения:



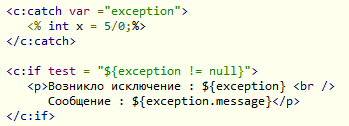
Редирект



**<c:import> -** импорт одной jsp страницы на другую.



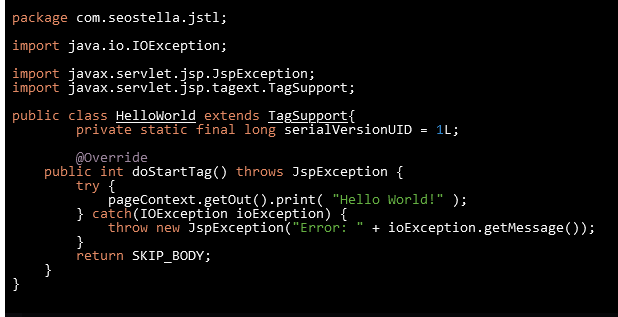
**<c:catch>** - обработка исключений в теле jsp.



**JSTL** позволяет **создавать свои кастомные теги**. Для этого нужно создать файл **/WEB-INF/tagName.tld.**



Класс, который обрабатывает тег



Чтобы им воспользоваться необходимо добавить taglib в jsp-файл:



Таким же образом работают и стандартные теги. Каждый тег имеет свой обработчик. Обработчик получает выходной поток, в который записывает необходимые данные.

**TOMCAT**

**Контейнер сервлетов –** программа, представляющая собой сервер, который занимается системной поддержкой сервлетов и обеспечивает их жизненный цикл. Он берет на себя такие функции как создание программной среды для функционирующего сервлета, идентификацию и авторизацию клиентов, организацию сессии для каждого из них.

Наиболее известные реализации контейнеров сервлетов:

* Apache Tomcat
* Jetty
* JBoss
* WildFly
* GlassFish

**Tomcat –** контейнер сервлетов. Предназначен для работы с рядом технологий Java EE, в частности, с JSP, сервлетами и рядом других.

В качестве контейнера сервлетов Tomcat выполняет несколько важных функций:

* создает среду для запуска сервлетов;
* задает параметры сессий;
* обеспечивает обмен данными между сервлетами и клиентами;
* проводит идентификацию и авторизацию клиентов;
* управляет метаданными.

**Основные каталоги:**

* **bin**: хранит различные скрипты, в частности, для запуска, перезагрузки и т.д.
* **conf**: хранит конфигурационные файлы, наиболее важным из которых является файл server.xml, который определяет основную часть конфигурации.
* **logs**: папка по умолчанию для хранения лог-файлов.
* **webapps**: папка, где хранятся собственно файлы приложений

Каждое приложение должно иметь папку **WEB-INF/classes**, в которой располагаются скомпилированные классы сервлетов.