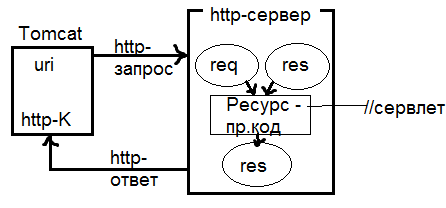
**ЛК 1. Программирование в Internet**



**Tomcat** – сервер., напис. в соотв JEпшE (опис. appl.server), но это урезанный вариант appl-server-а (под ним сервлет)  
**servlet = handler**

**Tomcat** – контейнер сервлетов, это С, обесп работу спецификаций JEE  
JEE – документация, сервер внутри webapps  
Tomcat вып в кач-ве службы Windows

**Структура папок Apache-Tomcat:**  
папка conf – Для конфигур С-а, там ряд файлов, позв настроить С:  
\* server.xml – порт, протокол, redirectPort, connectTimeout [мс] – допуск на подключ

\* web.xml – (в папке code) – опис. конфиг по умолч, sessionTimeout, web.xml есть (мб) в каждом приложении. Если его нет, смотрим общий, т.е. в папке code томката (по умолч).  
welcome-file-list – список стартовых страниц в порядке их перечисления – м. размещ в web.xml прил-я (приоритетнее)

\* tomcat-users.xml – юзеры, группы, роли. Поддерж basic, forms-аутент. Ресурсы Са мб защищены ролью (ресурс – uri, например)

□ **logs** – журналы – видим дату старта:  
 catalina.log – процесс загрузки сервера  
 localhost\_access.log – все, что кас. запросов  
 tomcat9\_stderr(stdout).log – все, что пишем

□ **bin** – исполняемые файлы (батники)  
 tomcat9w.exe – м. упр. работой Са

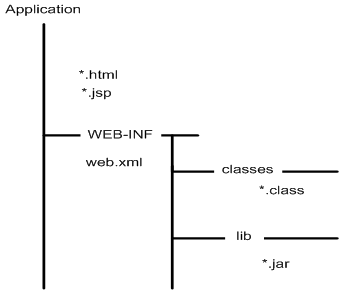
□ **webapps** – то что мы будем разраб

**Простейшее приложение:**  
webapps – SimpleApp – index.html  
localhost:48080/SimpleApp/  
будет наш html

**Servlet** – серверный web-компонент JEE-приложения предн. для обработки запроса, формир и отправки ответа  
**Servlet** – базовая единица, элементарное веб-прилож – java-класс, реализ. ин-с Servlet. Любой запрос к томкату встречает сервлет (явный или нет)  
\* есть в составе tomcat  
\* есть кот. сами разработали

**JEE** – не набор библ, а их описание! (это док), а реализацией заним-ся др. компании

**Структура папок Servlet:**

  
В папке classes откомпилированные классы объектов. Java-классы не компонуются, нах. по-отдельности. Мб папка WEB-INF – там ф., манифест с доп инфой

**Сервлет** – класс, реализ. ин-с Servlet (опосредованно : extends HttpServlet – это класс, кот реализ. ин-с Servlet)  
- есть к-р для иниц памяти и 3 метода:  
 \* **init** – 1 раз при созд объекта сервлета  
 \* **destroy** – вызыв сборщиком мусора  
 \* service

Init() и destroy() обесп **персистентность** (нек. св-ва S-a м. проиниц при его создании и при разруш S-a м. сохранить нек. св-ва), но сам S не гарант сохр. своего сост (stateless)

При обраб запроса, поступ на С, сраб сервлет. Если запрос 1-й, то new() servlet, init, service… (ему передаст req, res. Если серлвет уже создан, мб исп тот экземпляр или создан новый => stateless

1) Сервлет подразумев. переносимость на ур. байткода (пис. на Windows, получ. байткод, в Linux раб. без предкомпиляции)

2) Сервлет д. лежать в папках опр. структуры. Совместимость обесп и на ур. папок. Во всех appl-serv структура папок один. Перенесем папки WebSphere в Tomcat => все будет раб

Абсолютно **ЛЮБОЙ** запрос обраб сервлетом (встроен в Tomcat). Если запросим html, вызыв встроен. Ss, тот счит. html-файл и выводит на сторону К. (в любом appl-сервере в JEE)

**Способы взаимодействия Сервлетом др. с др. и др. компонентами:**

|  |  |
| --- | --- |
| переопред запроса (forward) | вызыв. service() др. сервлета \* взаимод. осущ в рамках 1 прил! (передает req, res ч/строку кот. он получил от др. Ss \* возврата нет! ответ отпр тот компонент, на кот мы сделали forward |
| переадрес запроса (redirect) | отпр res на К, там код 3хх и в заголовке ‘Location’ адрес нового рес, на кот. К обязан сделать новый запрос метод res.SendRedirect() \* м. нах в др. прил, сервере, хосте, где угодно \* если код не 307 и 308, повторный запрос всегда GET! (даже если был POST изначально), иначе тот же метод |
| непосред. запрос | м непоср отпр запрос к др. ресурсу (Ss в рамках service() исп Http-клиент. Делаем http-запрос на др. ресурс (в любом месте) и получаем сюда же ответ, обраб в рамках этого service() метода |

С-ма сборки **Ant** опис. XML-ом и сост. из target (цель), все цели именованы. Вначале в <project> указ. <default-copy>, т.е. его цель – вып. copy  
Когда дойдет в <target-copy> увидит depend? По стеку до init будет подним… Надо знать структуру папок  
А если делаем **Maven** (Смелову не нравится), все там по умолч, скрыто

**JAVA SERVER PAGES**

**JSP** – и есть сервлет в др. виде, определение то же  
JSP-стр = html-стр + директива ‘page’  
**JSP** – технология, которая позволяет создавать динамические веб-страницы. По сути это html-код с вкраплениями кода Java. Но это не стандартные html страницы, когда приходит запрос к определенной странице JSP, то сервер обрабатывает ее, преобразует в сервлет, генерирует код html и отправляет клиенту. В итоге после обращения пользователь видит просто html-страницу.

**JSP** – сервлет, написанный в другой форме. При первом обращении компилируется в сервлет. Является самодостаточной, может сама принимать запрос, обрабатывать и отвечать клиенту.

Существуют **3 конструкции**, которые используются в веб-программирование на Java:

* 1 сервлет и он встречает запрос, работает с бд и формирует ответ;
* 2 компоненты: сервлет и jsp: запрос встречается сервлетом, обрабатывается, дальше сервлет делает forward на jsp-страницу и jsp страница формирует ответ на сторону клиента. JSP выступает представлением в данном случае, а сервлет контроллером (походе на mvc); самая частая схема
* 1 jsp страница без сервлета – самодостаточна, может принимать запросы, обрабатывать и отвечать клиенту.

Для того чтобы сервер мог обрабатывать JSP страницы, необходимо движок JSP. У нас томкат является одновременно и веб-сервером для сервлетов и JSP.

**<%....%>** - для определения кода на Java.

- Если много статич текста -> лучше JSP  
- Если много динамики -> лучше Сервлет  
- Если MVC -> лучше JSP (=view, формир содерж) + Сервлет (=к-р, обраб запрос + forward на JSP)

Когда работает с jsp страницы, нам неявно доступны объекты для работы с ними в рамках jsp: request, response, pagecontext, session, out, application, config, page, exception и т.д.

**JSP:** мб директивы, выражения, объявления, действия

**Директивы:** - для установки условий, которые примен ко всей странице JSP  
- page (старт страницы) – c пом.атр.import можем импорт.пакеты или классы на jsp-стр  
- include (вкл др. страницу)  
- agrig (для библ тегов)

**Формы записи динам. контекста:**  
- объявления – позволяют определить метод, кот затем можно вызывать в скриплетах или JSP-выражениях. - <%! … %>; Например:

<%!

    int square(int n){

        return n \* n;

    }

%>

    <body>

        <p><%= square(6) %></p>

    </body>

//36  
- скриплеты – код Java - <% …%> - цикл for мб скриплетом  
- выражения – заключается между тегами <%= …. %> - вместо тега выведется содерж.

<%!...%> - на уровне класса, потом исп для др. динам эл-тов  
**Скриплет** – место, кот. превращ в метод класса сервлета (мб service)  
req, res нигде не опред, это авто- объекы в JSP, вверху есть Import  
**Выражение** – элем JSP, позв вычислить что-то и подставить значение

**Комментарии -**  <%-- … --%>;

**Комментрарии внутри скриплета:** <% // или /\*\*/ %>

**Стандартные действия jsp**

Веб сервер может иметь свой набор действий, определенных для jsp в виде тегов, распознаваемых компилятором jsp/ Также существует стандартный перечень действий, которые обязательно поддерживаются любым сервером.

**Библиотека тегов**

Это технология позволяющая создавать собственные теги, которые потом можно использовать в jsp-приложениях. Цель: разделить работу верстки и разработки.

***Последовательность действий:***

* Создать дескриптор библиотеки тегов (TDL) и поместить его в директорий приложения;
* Написать обработчики тегов – джава классы, генерирующие html текст, замещающий tdl-теги в выходные jsp страницы.
* Поместить на jsp-странице директиву taglib, управляющую на месторасположение дескриптора библиотеки тегов и задающую префикс (пространство имен) для имен tdl-тегов в данной странице.
* Добавить tdl-теги в jsp-страницу.

\* создать дескриптор библиотеки  
\* создать xml файл  
\* написать обработки библиотеки тегов (тег-хэндлеры), кот выполняют подмену тегов библиотеки на необх разметку  
\* использ директиву taglib  
\* Добавить новые разработанные теги на странице

Все теги: кот содержат тело внутри (jsp), и не содержат (empty)

WEB несколько расширяет понятие сервлет, т.к. там мб еще вебсокеты и др. компоненты.

….если версия веб сферы равна версии galssfish, и совместимость JVM. В рамках этого контейнера работает наше прил, т.е. обращаемся к функциям из контейнера. Конт предлагает ряд **системных объектов**, кот мы м исп в прцоессе работы. В томкат всего 4 типа таких объектов:

1) **контекст прил** – объект, кот создается при старте сервера (С стартует, ищет web.xml и формир контекст), сущ пока работает С, явл общим объектом для всего приложения. Самый долго играющий объект. В рамказ этого контекста сущ коллекция, кот создает память, в к-. м запис пары имя:значение (атриб) и извлек, м сделать setAttribute, removeAttribute. В кач значенпия часто использ ссылку на обхект. М создать объект к-то класса а ссылку его записать в контекст. Ест живая ссылка на объект и она будет хранится и garbage-мусорщик его не уберет. Те. контекст предоставляет нек память, кот хранится, пока работает С, если сами не удалим. Это первый уровень памяти

2) **session** – системный объект, созд для серии запросов, t м/которами не превыш sessionTimeout. Эта серия создается с пом куки, там session id. Каждый запрос помечается к какой сессии он относ. Если С не может определить сессию к к-й относ запрос, он создает новую сессию. Т.о. любой запрос пришедший на С, помечается к какой сесссии он относ. У сессии есть id (128 бит число) + память + коллекция (атрибуты имя:знач анал.) и эта коллекция сохр в рамках жизни сесии. СЕссия живет, пока жива серия запрсов. И для этой серии запросов м созранять нек данные (сохр состояние). Сессия – механизм сохранения состояния. Но когда сессия удалена, ссылка на объект пропадет и garbage cleaner его сожрет. Это второй уровень памяти (контекст для всех сессий общий => межсессионный обмен данных. Если хотим обмен между запросами => исп сессию). Крайние случаи – оч редкие запросы, для каждого своя сесиия, или очень частые запросы – одна сессия

3) **request** – живет в рамках запроса, тоже содрежит атрибуты и память. Это третий уровень памяти. Если хотим сохранить данные в рамках обработки 1 запрсоа, лучше в рамках объекта рекуест. В виде setAttribute запис данные, когда зарпос умрет, освободится ссылка и очищ.

4) **response** – не содержит атрибуты! живет столько же как и req, атрибуты не нужны. Когда приходит req, существует конвеейр его обрабоки – фильтр. Если надо данные передавать между фильтрами, лучше с пом атрибутов req.

**Объект ФИЛЬТР:**  
\* похож на middleware  
\* это серверный компонент, предн для предварит обработки запроса, для организации контейнера обработки запроса, «препроцессор запроса»  
\* обычный класс, реализ интерфейс Filter (тот оч похож на сервлет: метод init, destry, doFilter)  
\* фильтры созд при загрузке сервера. Они записаны в web.xml => вся инфа о фильтре хранится в контексте прил.   
- init() – при иниц фильтра, чтобы фильтр мог восст нек данные, кот он где-то сохранил (в БД, в файле…)  
- destroy() – чтобы мог данные записать, чтобы потом использ при восст  
init, destry – способы сохр состояния объектов между перезагрузками сервера  
- doFilter() – метод, вызывается контейнером сервлетов при поступлении запросов позв вызвать doFilter и передать парам: запрос, ответ, filterChain (Для образования цепочки фильтров)  
**Цепочка фи** – фильтры, привяз к 1 ресурсу (ресурс: сервлет, jsp) и они будут посл-но выполняться. Обычно фильтр завершается вызовом doFilter след. фильтра в цепочке. В конце цепочки последний фильтр вызывает service() cервлета, к-му передает req, res.

**Фильтр** – серверные компоенены для предварит обработки запросов прежде чем они попадут на конкр ресурс, созд при старсте сервреа

Они описаны в *web.xml*, там же мэппинг фильтров. Паттерн связывает фильтр с конкр ресурсом. Один и тот же класс фильтра мб исп в разных цепочках. Фактически это тупо middleware.   
Один фильтр вызывает другой, обрабатывая req, потом доходит до ресурса. Отрабатывает jsp/…, формир res и этот стек разворач в другую сторону. doFilter() - по стеку в глубь иду, на ресурсе делает петлю. На обратном пути можем обрабаывать уже res.

**Слушатели событий** – спец механизм обработки событий на стороне С. М создать классы, кот рализуют этот ин-с и записать инфу об этих классах в web.xml. тогда С при старте загрузит эти листенеры и мы можем обраб различ событий, связ с этими фильтрами. Есть события, связ с созданием/разруш контекста, измен атрибутов… Те же события можем подловить и обработать у сессии (создание, измен ее атрибутов). Чтобы создать обработчик таких событий, м создать класс , кот реализует 1/неск интерфейсов (смотря на какие события срабатывает). Чтобы С эти объекты мог поднять и зарегистрировать в контексте. Для этого пропишу их в web.xml C сразу поднимает фильтры и листенеры.

jstl – готовая реализация нек компонент.

М поместить собс данные в контекст. М записать параметры контекста кот потом м извлекать в программе с пом InitParam => обесп работу фреймворка

**JDBC** – java db community – прогр ин-с, обесп взаимод нашего прогр кода на Java с реляц БД. ODBC – самая быстрая, прародитель всхе таких технологий. Если прогр код на С++, его используем (либо OleDB).

Надо разделить ин-с на 2 компонента: одна – стандартная, общая для всего ПО, кот использ ин-с (.NET, Java пофиг), 2я – учит особености каждой СУБД (Драйвер бд – составная часть какой-то субд чаще, ее разрабатывают разработчики субд). В ADO.NET это называется ***провайдер***, но фактически это драйвер.

JDBC – jar-файл, ODBC – это dll-файл. Главный недостаток odbc – он не кроссплатформенный, в отличии от jdbc, кот написан на джаву, и работает на всех платформах, где работает jvm. Можно смело переностиь на уровне байт-кода.

JDBC API – набор программных ин-сов (поименованный набор сигнатур). А драйвер – классы, кот реализуют этот объект. **Connection** – Объект, предост нам драйвер, но этот объект реализует ин-с Connection, кот есть в jsbc. **Драйвер** – реализация этих ин-сов. ПОграничным явл объекь Connection, кот мы извлекаем из этого драйвера и начинаем с ним работать.

**Чтобы соединиться с сервером СУБД:**  
\* ip, где сервер или символический адрес (dns, netBIOS, hosts, gethostbyname помогут)  
\* порты 1521 oracle, 1433 mssql  
\* user name, password  
\* имя инстанса, к к-му мы коннектимся  
\* тип драйвера: тонкий или толстый (для oracle)

Все эти параметры в объекте Connection

Дальше м делать статические, динамические запросы и вызывать удал.процедуры, кот есть на стороне Сервера  
В оракл процедура м возвр только 1 строку, м еще здесь вызывать ф-ю в оракл

**update** – м формир рез набор (появился недавно) с пом секции output  
preparedstatement – то же самое executequery – в рез объекта класса resultset, кот мы м опреационным способом перебирать (это коллекция строк) с пом цикла, и обрабаывать их. Получаем как будто курсор для работы с рез набором

ссхема ромб

**JNDI** (java name and directory interface) – прилож работате с неск источниками д-х, к к. из них надо знать параметры соединения, мы не можем их зашивать в код, что с ними делать? М в property file (конфиг файл). Есть спец бд, позв хранить в себе такую инфу (в mssql – “Контроллер сети”, в oracle – “Active Directory”)

Эти с-мы наз **с-мами наименований**. В unix наз Directory. Протокол для таких бд – **L-DAV**.

В составе JEE входит **JNDI** – спец технология с пом кот 1) м написать ин-с для с-м наименований уже готовых, 2) JNDI дает набор ин-сов и спец. классов, с пом кот мы м разработать свою с-му наименований. В простейшем случае она выглядит как xml-файл, все к нему м обращаться и извлекать ту инфу, необх для соединений с другими источниками д-х. Там мб любая метаинфа по распределнной сети: имена юезров, сетевые пароли, коннекты. JNDI – та технология кот позволяет создавать такую службу наименования, кот м. расположить в своей сети (локальной). Обычно эти службы наименований – Сервер и какой-то заместитель у к-го есть копия этой службы наименования.