низкоуровневый код для обработки запросов (и ответов) это скучно неинтересно и не хотим каждый раз писать
что делать

1. библиотеки. берем реализацию, расширяем
2. высокоуровневая логика лежит в контейнере который низкоуровневое делает сам?

   это я буквально с лекции записывал...

(.war) специального вида архив специального вида приложения каким-то образом общается с контейнером
для языка общения (контракта?) -- API
специально для этих целей есть Servlet API
набор (интерфейсов, классов и тп) который позволяет приложению на одном языке разговаривать с тем контейнером где приложение размещено

maven3 -- система сборки

утилита mvn
в многих системах сборки (в т ч мэвэн) есть шаблоны сборки -- **архетипы** -- заготовки для определенного вида проектов
`mvn archetype:generate` -- в интерактивном режиме выберем и сделаем архетип
фильтруем по подстроке maven-archetype-webapp получаем 3 архетипа, выбираем
версию архетипа(берем последний)
указываем 3 параметра артефакта:
в экосистеме мэвэн проекты библиотеки архетипы? плагины и т п после того как они упакованы и размещены в нем называются **артефактами**
артефакты характерезуются 2-3 параметрами:
groupId и **artefactId**
как фамилия имя
хорошая идея использовать в качестве `groupId` название пакета
и **версия** -- понятно что
с точки зрения мэвэна проект это набор файлов (src) и файл pom.xml -- project object model?
pom.xml -- **дескриптор модели нашего проекта** ,
который описывает где исходники расположены, где зависимости на какие библиотеки что нужно делать на какую фазу какие плагины подключить и т п
в pom.xml прописываем зависимости в т ч servlet api
`scope:provided` -- зависимость нужна только на этапе компиляции, потому что далее ее будет предоставлять сам контейнер

<написать про структуру pom.xml ?>

после того как меняем pom.xml нажимаем в идее reload

webapp -- папка с не-исходниками? -- статическими веб ресурсами

.war файл -- zip архив специального вида, в который соберется наше приложение
Контейнер должен прочитать этот веб архив, и взять оттуда настройки конфигурации и наши сервлеты?

==web.xml ==-- **дескриптор вебприложения**

хуячу темы через гугл и доки, а потом смотрю лекцию и записываю че упустил

Servlet API

Servlet API -- программный интерфейс который стоит между нашим веб приложением и контейнером где мы его размещаем -- с лекции

`try` с ресурсами -- чтобы потом закрылся
кстати в 4й лекции новых лет он уже говорит что *именно закрывать* не надо, у нас же keep-alive, так что просто `flush` -им (или контейнер сам флашит?)
в одном контейнере можно запускать несколько .war-ников (веб приложений)

сессия -- работает через куки (на сервере мапа)
сервлеты на урлы мы мапим в web.xml

~~(взято с javarush):~~
~~**Java Servlet API** — стандартизированный API, предназначенный для реализации на сервере и работе с клиентом по схеме запрос-ответ.~~
~~**Сервлет** — это класс, который умеет получать запросы от клиента и возвращать ему ответы. Да, сервлеты в Java — именно те элементы, с помощью которых строится клиент-серверная архитектура.~~
UPD: #todo перевести описание ниже на русский

Документация :
A **servlet** is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol. тут
These are known as life-cycle methods and are called in the following sequence:

1. The servlet is constructed, then initialized with the `init` method.
2. Any calls from clients to the `service` method are handled.
3. The servlet is taken out of service, then destroyed with the `destroy` method, then garbage collected and finalized.

интересный факт:
мапить можно и через аннотации но мы на тех практиках просто не хотели так делать

Что вернет если не реализован doPost ?
ответ:

```
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException
{
    String protocol = req.getProtocol();
    String msg = lStrings.getString( key: "http.method_post_not_supported");
    if (protocol.endsWith("1.1")) {
        resp.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED, msg);
    } else {
        resp.sendError(HttpServletResponse.SC_BAD_REQUEST, msg);
    }
}
```

выжимка из документации к HttpServlet :
сервлеты параллельны:
"Servlets typically run on multithreaded servers, so be aware that a servlet must handle concurrent requests and be careful to synchronize access to shared resources. Shared resources include in-memory data such as

instance or class variables and external objects such as files, database connections, and network connections."
doXXX:
гарантируется что `init()` и `destroy()` запустятся ровно один раз за жизнь сервлета
но функции `doXXX()` будут вызываться типа параллельно, поэтому нужно думать об общих ресурсах
service : Receives standard HTTP requests from the public `service` method and dispatches them to the `do`*XXX* methods defined in this class -- его нам обычно не зачем менять

tomcat

-- servlet-контейнер, очень популярный
вики:
Tomcat (в старых версиях — **Catalina**) — контейнер сервлетов с открытым исходным кодом. Реализует спецификацию сервлетов, спецификацию JavaServer Pages (JSP) и JavaServer Faces (JSF). Написан на языке Java.
Tomcat позволяет запускать веб-приложения и содержит ряд программ для самоконфигурирования. Tomcat используется в качестве самостоятельного веб-сервера, в качестве сервера контента в сочетании с веб-сервером Apache HTTP Server, а также в качестве контейнера сервлетов в серверах приложений JBoss и GlassFish.

Catalina for Tomcat

Catalina — контейнер сервлетов Tomcat, который реализует спецификацию сервлетов Servlet API. Servlet API является основой для всех остальных технологий Java, касающихся Web и дает возможность динамически генерировать любой веб-контент, используя разные библиотеки, доступные в Java. Архитектором Catalina являлся Craig McClanahan.

темная сторона луны

#todo : gzipFilter из домашки

gzip — утилита сжатия и восстановления файлов

doFilter
The `doFilter` method of the Filter is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.

#todo что делает writeListener.notify(); и что теперь работает по другому
например в статике мы всегда отправляем файл через File.copy(file, outputStream), но при этом видимо мы при помоши фильтра умудряемся его сживать если позволяет пользовательский агент

ServletResponse

To send binary data in a MIME body response, use the `ServletOutputStream` returned by `getOutputStream()`. To send character data, use the `PrintWriter` object returned by `getWriter()`. To mix binary and text data, for example, to create a multipart response, use a `ServletOutputStream` and manage the character sections manually.