|  |  |
| --- | --- |
| JDK 1.8  IntelliJ IDEA  Maven  Git  Jetty  freemarker | Java Development Kit 8  среда разработки приложения на Java  система сборки проекта  система контроля версий  библиотека для web сервера  шаблонизатор |

Maven мы будем с вами в основном выкачивать зависимости. То есть выкачивать библиотеки, которые нам понадобятся в добавок к стандартным библиотекам Java. Частично будем собирать проекты Maven, но я думаю, что все‑таки основная работа по сборке у нас будет происходить в IDEA.

При помощи 2‑х библиотек Jetty мы будем строить веб­‑сервер.

Понадобиться freemarker — простой, но удобный и надежный шаблонизатор.

|  |
| --- |
| Распакуйте архив в инсталляционную директорию.  Windows: C:\ProgramFiles\maven\  Linux: /opt/maven  Установите переменную окружения  M2\_HOME = инсталляционная директория  Установите переменную окружения PATH  Windows: в переменной PATH добавьте к списку директорий строку %M2\_HOME%\bin  Linux: можно добавить строку export PATH=$PATH$M2\_HOME/bin в файл /etc/profile  Запустите в командной строке mvn -version |

Maven — это система контроля зависимости и сборки проектов. То есть это приложение, помогающее вам выкачивать нужные вам версии библиотек и собирать проект. Значит у Maven есть большой центральный репозиторий. Вы можете набрать Maven в поисковике и первой же ссылкой получите ссылку на этот репозиторий. Вы можете посмотреть какие в нем библиотеки лежат. Их там невероятное количество. Все что нам может понадобиться в Maven репозитории в центральном есть. Наше использование Maven сведется к тому, что мы будем говорить, что нам нужно выкачать. Maven нам будет выкачивать сам. Вам понадобиться интернет, чтобы выкачивать библиотеки Maven как минимум при первом старте, при первом открытии вашего проекта в среде разработки. Настройка чуть сложнее чем все предыдущие, хотя ничего особенно страшного нет, нужно просто проставить несколько переменных окружения. В Linux и в Windows будет немного отличаться, но я думаю, что любой уверенный пользователь этих операционных систем разберется. И в конце, чтобы понять, что вы все сделали правильно, вы можете просто в командной строке набрать mvn -version и Maven вам покажет какая версия была установлена. Если при запуске командной строки вы видите mvn не найден, значит что‑то у вас пошло не так. Последние версии IntelliJ IDEA содержит в себе Maven уже встроено. Все эти действия, которые на этом слайде перечислены, нужны, если вы собираетесь собирать проекты вне IDEA. В принципе я вам рекомендую все это проделать. Если вдруг сразу не получилось, будем использовать то, что у вас в IDEA. Если понадобиться больше, потом до настроите.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14. | Project Object Model — pom.xml  <groupId> адрес проекта </groupId>  <artefactId> название приложения  </artefactId>  <version> версия приложения </version>  <dependencies>  <dependency>  <groupId>org.eclipse.jetty</groupId>  <artefactId>jetty-server</artefactId>  <version>9.3.0.M1</version>  <dependency>  </dependencies> |

Итак, если вы успешно проделали все предыдущие пункты, то у вас на рабочей машине уже установлена JDK нужной версии, есть среда разработки, стоит Maven, и вы готовы уже начать создавать проект. Я вам предлагаю первым делом зайти на сайт JetBrains, разработчика IntelliJ IDEA, и почитать как именно нужно создавать проекты под управлением Maven. После того, как вы пройдете все шаги, которые вам рекомендует в инструкции, у вас получится пустой проект, и в нем будет файл с именем pom.xml (Project Object Model). В нем вы будете указывать какие именно библиотеки вы хотите подключить к своему проекту. И как именно вы хотите свой проект собирать, если вы хотите его собирать отдельно не из IDEA. В каждом pom файле вы найдете блок с описанием вашего проекта. Он будет содержать groupId, artefactId и version. И в этот же pom файл вы сможете записывать зависимости, которые вы хотите, чтобы у вашего проекта были на библиотеки других разработчиков. И при включении зависимости вы также должны указать groupId, artefactId и version того, что вы хотите выкачать. На слайде 8‑14 я привел пример зависимости на библиотеку Jetty, на Jetty сервер с версией 9.3.0.M1. Естественно время идет и версии будут у библиотек расти. Те версии, которые я указываю в своих слайдах, это те версии, которые сейчас у меня в примерах, которые у меня работают. В принципе вы всегда можете работать с теми версиями, которые есть у меня. Если вам нужно только библиотеки для примеров. Если вы в дальнейшем планируете разрабатывать проект дальше курса и использовать его как‑то еще в добавок тому, как мы рассказываем в курсе, то вам хорошо бы было следить за версиями библиотек и расставить зависимости на последнее.

1.3.3 Jetty

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12. | Java based http server  создание динамических страниц  пересылка статических файлов (html страницы, картинки…)  <dependencies>  <dependency>  <groupId>org.eclipse.jetty</groupId>  <artefactId>jetty-server</artefactId>  <version>9.3.0.M1</version>  <dependency>  </dependencies> |

Итак, первое что нам с вами понадобиться это библиотека Jetty. Перед тем, как рассказать про Jetty я хочу рассказать пару слов о том, как у нас вообще будет устроена работа с пользователем. В этой части курса, части посвященной бекэнду, мы с вами будем заниматься непосредственно разработкой сервера, того, что реагирует на запросы пользователей. У нас будет то место нашего веб‑приложения, которое отвечает за обработку запросов за формирование ответов пользователю на основе данных, которые пришли от пользователя, а также на основе нами собственно вычисленных данных или данных, взятых из базы. Мы с вами существенно облегчим себе работу тем, что не будем писать свой собственный клиент. Клиент у нас — это браузер. Вы будете отдельно программировать свое приложение во второй части нашего курса так, чтобы браузер показывал пользователю красиво и удобно то, что мы ему вернули. Наша задача будет обработать запросы и сформировать ответы. Упрощение, которое я предлагаю сделать в данном случае, — это кроме того, что мы не пишем свой собственный браузер, то есть в качестве клиента мы используем уже готовый браузер, мы с вами также не будем писать протокол общения между клиентом и сервером, то есть в качестве протокола у нас будет http запрос. Та часть нашего сервера, которая непосредственно будет работать с браузером и с его запросами, то есть работать в смысле принимать эти запросы — это будет некий http сервер. В качестве этого http сервера я вам предлагаю использовать Jetty. То есть смотрите, наша с вами задача будет писать обработчики запросов пользователя. То, как именно эти запросы уходят от браузера пользователя и прилетают к нам, мы с вами писать не будем. То есть ни клиент, ни протокол общения, ни непосредственно уровень приема запросов мы с вами разрабатывать не будем. Вместо них будем использовать уже готовое решение. Нам и без этого достаточно что сделать. Та часть нашего сервера, которая будет принимать запросы пользователя, запросы из разных браузеров, это и есть http сервер, в нашем случае Jetty. Первое что вам нужно будет сделать для того, чтобы начать общаться с браузером — это подключить себе его в ваше приложение. <dependency> на него я уже показывал. Вам нужно будет открыть pom файл вашего проекта, вставить <dependency> как есть, может быть зайти на сайт Maven и посмотреть какая сейчас версия последняя, поменяйте версию, посмотрите. Те версии, которые у меня здесь, я их протестировал. Если вы выставляете свои версии, там могут начаться какие‑то особенности. Например, какие‑то классы могут показаться уже deprecated. То есть вам придется самостоятельно согласовывать версии. С первого раза я вам предлагаю поставить мои версии, а потом, если вы выходите уже на новый уровень разработки, включать следующие. После того, как вы это включили, в вашем приложении появится возможность использовать Java бейст http сервер, то есть библиотека, которая вашему приложению позволяет обрабатывать запросы, которые в ваш сервер будут поступать из браузера. И наша задача в результате будет сведена к тому, что мы должны будем написать у себя в коде обработчики для этих событий. Мы с помощью Jetty. Jetty нам просто скажет унаследуйте класс от определенных базовых типов, реализуйте некий набор функций и в том случае, если пользователь что‑то захочет от вас получить, то его запрос приведет к вызову функции, которую вы напишите. Это и является обработчиком. Еще раз. У нас клиент серверное приложение, мы самостоятельно с вами клиент не пишем, мы программируем для клиента в другой части курса. Клиент этот, браузер, отправляет запрос по протоколу, который мы тоже не пишем. Для того чтобы понимать этот протокол и этот запрос у нас с вами в составе сервера будет хетпати библиотека Jetty, она получит эти запросы и по виду этих запросов поймет какие именно функции вызвать в нашем приложении. В эти функции она заодно еще отправит request от пользователя. request — это уже название класса. В этом request будет информация о запросе. После того как мы из этого request всю нужную информацию возьмем и обработаем ее. Например, если будет request на авторизацию или регистрацию пользователя. Сделаем нужные действия и в ответ отправим response. response — это тоже название класса. Что значит отправим? Значит, что мы из своей функции вернем response, либо мы его поменяем и Jetty сама поймет, как его поменяли. В ответ нам нужно будет сказать код результата и строку. Это строка, которая в браузере будет превращена в страницу, которую браузер отработает. Еще раз. Если вы работали с http серверами, такими как Apache или Nginx, то вы в общем то представляете что это. Вы поднимаете некий сервис. Он обрабатывает запросы. По запросам возвращает в случае Apache и Nginx статический файл. Jetty тоже умеет возвращать статические файлы. Вы можете настроить Jetty так, что она будет запросы на статику перенаправлять в некоторую директорию. В этой директории искать файлы с соответствующим названием и возвращать их. Кроме того, что Jetty умеет возвращать статику, она также умеет, собственно, динамические страницы. Это то самое, о чем я сейчас рассказывал. Динамические страницы — это страницы, созданные на основе запросов. И наша с вами задача будет написать обработчики для создания динамических запросов. Сама обвязка, которая приводит к вызову этих обработчиков, эту часть мы поручим Jetty.

1.3.4 Jetty

|  |  |
| --- | --- |
| Server:  connector  handler(s)  ThreadPool | принимает http запросы (request)  возвращают response  pool потоков для обработки запроса |

Теперь, когда мы уже понимаем зачем нам библиотека Jetty и смогли ее подключить в наш проект, несколько слов о том, как она работает и как она устроена. На сайте Jetty есть статья об устройстве и работе Jetty. И там есть эта замечательная диаграмма, которая есть на этом слайде. К ней подпись, что это взгляд на Jetty с высоты птичьего полета. Очень в крупные блоки собраны части, из которых Jetty состоит. Нам для ознакомления вполне достаточно такого взгляда с высоты птичьего полета. Если начать разбирать внутренне устройство, первое о чем нужно сказать — это Connector. Эта та часть Jetty сервера, которая непосредственно принимает запрос от браузеров, от разных пользователей. То есть если пользователь что‑то запрашивает, он должен задать адрес, по которому он запрашивает, адрес и потом документ, который он запрашивает. После того, как он сформировал запрос, запрос будет понят браузером и отправлен на машину, которой этот адрес принадлежит. На этой машине на определенном порту, стандартный порт для htttp запросов 80, должен висеть обработчик, http сервер, который весит на порту и слушает запросы, которые на этот порт приходят. Jetty сделает так, что на наш сервер при старте тоже будет цепляться к некому порту. Цепляться и слушать запросы. И вот та часть, которая непосредственно слушает запросы пользователя, которая работает с портом и с запросами — это connector. После того, как данные в connector, то есть мы сформировали некий набор запросов. Дальше процесс исполнения приводит к тому, что мы вызываем, то есть мы должны неким образом этот вызов должен быть понят Jetty, либо превращен в отдачу статических данных обратно пользователю, как ответ на тот запрос, который от пользователя пришел. Либо мы должны обработать этот запрос каких‑то из наших обработчиков самостоятельно записанным. Для того, чтобы обработать запрос у Jetty есть TreadPool. Это часть сервера, в которой лежат заготовленные потоки. Когда мы дойдем до многопоточности, эта часть будет чуть более понятна. Пока что представьте себе это как остановленные линии конвейера. То есть у вас завод, в нем много стоящих неработающих линий конвейера и когда приходят пользователи, запрос пользователя сделать что­‑то, Jetty выбирает какую‑то из этих линий, в данном случае поток из пула, запускает его и в нем исполняет задачу. ThreadPool — это набор заготовленных элементов, которые могут исполнять что‑то. Взяв запрос и взяв поток из пула сервер может в нем вызвать непосредственно обработчик. То есть Hendler — это то, что непосредственно обработает запрос. Это может быть опять же запрос к файловой системе, которая приведет к отдаче статического файла, либо это запрос к нашему коду для того, чтобы в нашем коде сформировать ответ и вернут его пользователю.

1.3.5 Jetty

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19. | public class Main {  public static void main(String[] args)  throws Exception {  Frontend frontend = new Frontend();  Server server = new Server(8080);  ServletContextHandler context =  new ServeletContextHandler(  ServeletContextHandler  .SESSIONS);  server.setHandler(context);  context.addServelet(  new ServeletHolder(frontend),  '/authform');  server.start();  server.join();  }  } |

После того как мы подключили Jetty, разобрались зачем вообще она нам нужна и из чего и каких больших блоков она состоит и как работает, самое время уже посмотреть участок кода, который позволит нам Jetty запустить. Здесь я привел класс Main и в нем функцию main(). Название класса Main произвольное. В принципе функция main() может быть в любом из Java классов. А вот сигнатура main() уже обязательная. То есть вы хотите создать точку входу в вашу приложение, вы должны написать 2 и передать аргумент. В этом примере первым делом я создаю обработчик. Frontend (4), который здесь указан, — это мой собственный класс. Как я уже говорил, чтобы создать объект в Java нужно вызвать new. Что такое Frontend я вам покажу чуть позже. Здесь пока рассказ о том, как запустить сервер. Server (6) — сервер Jetty. Это основной класс библиотеки Jetty. И при старте сервера мы должны еще ему сказать самую важную для него информацию, это то, на каком порту он будет стартовать. В данном случае в этой строке я создаю объект Jetty сервера и говорю ему, что он должен работать на порту 8080. Это обычный тестовый порт для разработки приложения. Дальше мне понадобится создание сервлет‑контейнера ServletContextHandler и в него мне нужно будет передать то, что будет обрабатывать запросы. То есть, когда я говорю здесь frontend (13), это объект, который я ранее создал. Я говорю, что этот обработчик должен обрабатывать запрос, который придет на запрос документа authform (14). Кога пользователь захочет обратиться к моему серверу, он наберет адрес. В моих примерах это будет localhost. Он наберет порт. То есть запрос от пользователя может выглядеть следующим образом. У нас будет http://localhost:8080. И после этого пользователь еще должен будет написать документ, который он хочет запросить. Это будет authform. Если пользователь все это сделает и если сервер запущен в таком виде, то запрос от пользователя придет внутрь обработчика объекта класса Frontend.

1.3.6 Jetty. Сервлеты

|  |  |
| --- | --- |
| Servlet  ServletContainer | класс расширяющий возможности сервера  как aplet, только на web сервере  java решение для создания динамических страниц  объект, который обрабатывает http запрос (request) и возвращает ответ (response) — html страницу  часть web сервера, содержащая сервлеты, связывает URL с сервелетом |

Дальнейший разбор работы с Jetty и с обработчиками невозможен без упоминаний сервлетов. Потому что на предыдущем примере в общем то я уже сказал слова сервлет и сервлет‑контейнер. Мы с вами, используя Jetty, разрабатывая веб‑сервер, будем писать сервлеты. То есть наша задача использовать готовый веб‑сервер и объяснить ему какие сервлеты для чего использовать. Название сервлет не очень удачно. Изначально было 2 родственных понятия сервлет и апплет. То есть апплет появился раньше, как некая сущность, расширяющая возможности браузера. И так как апплет был на фронтендной стороне, то решили что давайте на серверной стороне тоже будем называть то, что расширяет возможности сервера, словом сервлет. Если вы настраивали допустим Apache, который для динамических страниц PHP, то примерно можете представить себе, что такое сервлет. Apache задуман как для отдачи статических страничек. Но его можно расширить неким скриптовым языком, например, PHP, который позволит создавать странички динамические. И на каждую страничку вы писали файлик, собственно, название файла .php. И вот это прямой аналог сервлета. Jetty сервер нам дает возможность отдавать статические документы и он вообще нужен для того, чтобы принимать запросы. А мы еще можем ему в добавок дать возможность обрабатывать запросы динамические. Пользователю не важно, как мы запросы обрабатываем. Это мы у себя знаем, что они динамические. Пользователь просто присылать нам некоторые данные. Он присылает нам request. А мы для этих request можем писать код, который не просто отдаст ему готовую страницу, а который на основе данных этих request залазит в базу, соберет какие‑то материалы и отдаст ему. Это как раз и есть действие сервлета. У нас сервлет — это обработчик, привязанный к некому запросу. А запрос со стороны пользователя — это запрос документа. Пользователь запрашивает документ, а мы знаем, что этот документ на самом деле не документ в файловой системе какой‑то файл, а запрос на динамическое создание контента. Мы с вами будем на каждый пользовательский запрос писать сервлет либо на группу запросов и в нашем понимании сервлет это будет наследник некого класса, который будет содержать функции, которую мы напишем для того, чтобы запрос обработать. После обработки мы должны сформировать страницу так, как будто Jetty прочитал эту страницу из файла. И пользователю, и Jetty на самом деле все равно откуда взялась строка, которой мы даем в качестве response. Эту строку можно получить чтение файла, а можно получить динамически, например запросом к базе. И в response и том и в другом случае у Jetty будет строка. Наша задача по запросу пользователя создавать строки для разных запросов разные и эти строки в виде response пользователю отдавать. И вот за то, чтобы формировать динамический контент пользователю мы будем использовать сервлет. На этом слайде я упоминаю еще севрлет‑контейнер. Это та часть сервера, которая содержит мапинг документа, который пользователь запрашивает на сервлет, который этот документ обрабатывает.

1.3.7 Jetty

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7. | <dependencies>  <dependency>  <groupId>org.eclipse.jetty</groupId>  <artefactId>jetty-webapp</artefactId>  <version>9.3.0.M0</version>  <dependency>  </dependencies> |

Перед тем, как начать использовать сервлеты, нам эти сервлеты нужно подключить к себе. На этом слайде я привел зависимость. Эту зависимость вы точно так же, как зависимость на Jetty сервер, должны включить в блок с зависимостями своего pom документа. После того, как вы это сделаете, среда разработки спросит у вас хотите ли вы выкачать версию. Соглашайтесь, она вам выкачает добавку к Jetty, которая непосредственно работает с сервлетами.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20. | public class Frontend extends HttpServlet {  private Sring login = '';  public void doGet(  HttpServletRequest request,  HttpServletResponse response)  throws ServletException,  IOException {  ...  }  public void doPost(  HttpServletRequest request,  HttpServletResponse response)  throws ServletException,  IOException {  ...  }  } |

После того, как все это будет сделано, вы можете уже написать вы можете написать свой собственно первый сервлет. Это тот самый класс Frontend, который я уже упоминал. Это то самое название, которое было у нас в примере одного из предыдущих шагов. И как я обещал он будет наследовать некий базовый класс. В нашем случае это HttpServlet. Вот как раз HttpServlet — это класс, который лежит в библиотеке, который вот только что с зависимостями выкачали себе. В каждом HttpServlet можно разместить 2 функции doGet и doPost. Функции, которые будут вызваны Jetty в тот момент, когда на этот сервлет придет GET или POST запрос. Смотрите еще раз. У нас есть Jetty сервер. Jetty сервер цепляется к некому порту и слушает запросы. Внутри Jetty сервера есть настройка, которая говорит с какой директории брать статические документы. И есть настройка, которая позволяет привязывать запросы некого документа к некоторому сервлету. Если вы вернетесь к предыдущему примеру, вы увидите, что сторону фронтедн я привязал к запросу на AU форм. То есть это некая страничка. Если пользователь ее запросит. Скорее даже так. Пользователь запросит ее при отправке формы. То есть пользователь запрашивает наш сервер. В индексе получает статический документ с формой. В форме заполняет поля авторизационные. И в форме указано куда пойдет запрос, на какой адрес. И это адрес как раз AU форм. После того как пользователь форму заполнил и нажимает отправить, браузер подсоединяется к нашему серверу, тому порту на котором висит Jetty и слушает. Jetty находит в потоке пулов свободный поток, поручает этому потоку обработать запрос и в зависимости от того как именно вы отправляли форму get или post запросом в этом потоке будет вызвана либо функция doGet, либо функция doPost. В этих функциях как вы можете видеть нам пришло еще 2 переменных. Это request и response. И в doGet, и в doPost. И в request нам Jetty сообщает все те переменные, которые нам могут понадобиться пользователи. Те, которые он сам им передает, и те которые является его переменными окружения и в response мы с вами должны будем записать ответ, который мы хотим пользователю отдать. То есть в response мы проставляем ключ результата. И в response мы пишем строку, которая потом превратится в страницу у пользователя. Браузер получит эту строку и отобразит ее как страницу.

1.3.8 Шаблонизатор

Картинка

В заключении этого урока я хочу вам рассказать еще об одной библиотеке, которая упростит нашу работу по созданию страниц. Как я уже упоминал раньше пользователь от нас ждет response. Браузер пользователя джет response. В response мы пользователю даем строку. Браузеру все равно откуда взялась строка. Прочитана она из файла. То есть что такое отдать пользователю текстовой файл или статическую html‑страницу — это прочитать содержимое текстовой страницы и отдать ее в браузер. То есть в общем‑то браузеру в любом случае прилетает строка. Просто мы знаем, что в этой строке страница. И браузер понимает, что это страница. А можно эту строку взять и сформировать программно. Не читать из файла, а прям взять и целиком написать у себя в приложении строку, захаркодить ее и собственно отдавать из класса. Это можно так делать, но это будет не удобно. При работе с вашими страницами на самом деле, при создании динамического контента в большинстве случаев не нужно формировать целиком всю страницу. Вам нужно в ней поменять какой‑то набор полей. То есть написать, пользователь авторизуется, и вы должны после авторизации написать ему «Здравствуй» и указать его имя. Та часть страницы, которая содержит заголовок, кодировку, разметку, указания на стили — она не будет меняться от запроса к запросу. У вас меняться будет только слово имя пользователя. И все. И для того, чтобы не держать у себя в коде всю страницу со всеми разметками и с строками, которые вы не собираетесь менять, есть шаблонизатор. Это библиотеки, позволяющие вам совместить данные, полученные из самого файла, и данные, полученные каким‑то динамическим путем. То есть у вас, с одной стороны, есть файл, обычный файл, который лежит на диске, то есть в этом файле вы при старте вашего сервера говорите где эти файлы искать. С другой стороны, у вас есть поля, которые вы пользователю хотите вернуть. Библиотека, в нашем случае FreeMarker позволяет вам совместить плюсы от файла и значение полей. Все это вместе перемешать, пользователю вернуть уже сформированную страницу. В принципе совсем не обязательно использовать ее именно для генерации страниц. То есть FreeMarker, он может вам понадобиться, когда вы просто захотите формировать строки, часть которых создано статически и лежит в файликах и часть которых вы создадите динамически.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9. | <html>  <head>  <title>Пример шаблона</title>  </head>  <body>  <p>Имя пользователя: ${userName}</p>  <p>Id пользователя: ${userId}</p>  </body>  </html> |

Такая строка может быть. Этот файлик лежит на диске. В этом файлике у вас указаны поля, которые вы хотите динамически подменять. Вот такая нотация ${имяПеременной} она довольно распространена и относится не только к FreeMarker. Вы, используя другие шаблонизаторы, тоже ее встретите. Вы создаете страницу. В странице пишите места, которые должны быть заменены при формировании уже результирующего документа. После того, как вы свой код включите в библиотеку, вы сможете взять файлик, взять переменные, совместить их, получить строку, положить ее в response и отдать ее пользователю. Подробности использования FreeMarker и Jetty и использования севрелетов в Jetty я покажу при разборе примера

2.1.2 Задача

|  |
| --- |
| Сессия пользователя  1. Регистрация  2. Авторизация  ~~3. Игра~~  4. Выход |

Итак, давайте разберем задание, которое нам предстоит в этом уроке сделать. Я предлагаю вам в качестве задания написать веб сервер, на котором нужно будет авторизоваться. Заготовка, которая нужна для любого дальнейшего использования этого веб‑сервера. Как бы вы не планировали его использовать в качестве игрового сервера или в качестве CRM‑системы. В общем не важно. Даже если вы пишите собственную социальную сеть. Вам в любом случае понадобиться механизм регистрации и аутентификации пользователя. Наше задание с вами сейчас — это сделать первый прототип авторизации на нашем сервере. Как видно на слайде — первая наша задача будет сделать регистрацию пользователей. То есть то, что позволит пользователю сказать, что он хочет нашим сервисом в дальнейшем пользоваться, чтобы система узнавала его. Авторизация подразумевает, что пользователь уже есть в нашей системе, что мы его уже когда‑то запомнили. И он хочет нам представиться. Дальше следует некий пункт. Содержательное взаимодействие пользователя с сервером. И это мы пока с вами в этом занятии не будем. То есть третья часть у нас остается пока не написанной. В эту 3-тью часть можно будет вставить какое‑то наполнение, под которым я подразумеваю некое взаимодействие с пользователем, которое вы в дальнейшем напишите. Четвертый момент — это выход пользователя из системы. Когда он говорит я в общем‑то уже пользоваться вашим сервисом и хочу, чтобы все остальные пользователи, если запросят онлайн я или нет, уже знали, что я вышел.

|  |
| --- |
| Упрощения  1. Без записи в базу данных  2. Без интерфейсов  3. Потоками управляет Jetty  4. «Потокоопасные» карты |

В процессе работы нам с вами, так как мы только начинаем разработку придется сделать некие упрощение системы. Не все мы будем делать полноценно и сразу. Мы вместо некоторых сервисов сделаем пока заглушки. И самое важное здесь это то, что мы пока ничего не будем писать в базу. Базу мы разберем четь позже. У нас просто будет запись в runtime. То есть вы подняли сервер, пользователь пришел, зарегистрировался, залогинился, мы его узнали. Потом мы погасили сервер и запустили снова. Он приходит, мы уже его не знаем, потому что у нас runtime информация была потеряна при старте. Чтобы такой потери не происходило как раз и нужны базы. Мы их разберем позже. Пока что у нас никаких баз не будет. Пока что мы с вами не будем делить сервер на интерфейс и реализацию. Это некая архитектурное упрощение системы. Оно подходит для небольших проектов и пока для первого прототипа нам тоже подойдет. Хотя в дальнейшем я буду настаивать на том, чтобы для всех сервисов, которые мы пишите вы обязательно производили разделение и особенно будет понятно зачем это нужно, когда мы с вами подойдем к тестированию. У нас 3-ий пункт в упрощениях. Мы будем использовать в работе потоки, которые нам даст Jetty. Когда я рассказывал про Jetty, я говорил, что у нее есть отдельный пул потоков, сравнивал пул потоков с некой лентой конвейеров. Когда есть некая задача вы можете запустить конвейер, обработать ее, остановить конвейер. Под этими конвейерами потоками я подразумеваю что это даст нам Jetty. У нас будет поток Main, но вы в нем произведем всю инициализацию, которая нам нужна, а потом просто скажем просто подождать. Так как тема многопоточности пока еще не является для вас, ну мы просто не можем к ней приступить, потому что еще только в самом начале пути, то нам придется здесь использовать поток безопасной карты. То есть в том виде, в каком мы с вами напишем сервер, его можно в общем то использовать в продакшн тоже, хотя если вы все таки планируете это делать, то вам обязательно нужно будет послушать тему про потоки, иначе вы получите некоторые странности в поведении сервера, если к вам придет более одного пользователя одновременно. Веб‑сервер очевидно должен выдерживать нагрузку больше, чем один пользователь в один промежуток времени.

2.1.3 Задача

|  |
| --- |
| Регистрация  Frontend  1. Страница с вводом: логин, email, пароль  Backend  1. Сервлет обработки запроса  2. Класс AccountService с методом регистрации  3. Класс UserProfile с полями: логин, email, пароль  4. Карта логин -> UserProfile |

И теперь разберем страницы, которые нам с вами понадобятся. Когда на прошлом занятии я говорил вам про сервлеты в общем то я и говорил про эти самые страницы. То есть вспоминайте материал прошлого урока и на нем мы говорили, что для того чтобы кастомно каким‑то образом нестатически обработать запрос пользователя мы должны к этому запросу подключить некий сервлет. Если пользователь у нас просто запрашивает некую статическую страницу, текст или картинку, то в общем то нам просто можно отдать ему файлик с содержимым и все. А если пользователь хочет от нас некого действия. Например, как в случае с регистрацией он нам присылает данные, где среди этих данных вычисляет свое имя, email, пароль. И пользователь хочет, чтобы мы не просто вернули ему некий документ, а чтобы мы произвели у себя на сервере некие действия, которые приведут к запоминанию нами его информации. Вот это запоминание, код, который будет производить обработку данных о пользователе, этот код будет содержаться в сервлетах. На фронтенде пользователю мы должны предоставить страницу. Страница, на которой будет форма и в этой форме он сможет заполнить логин, емайл и пароль. Если вы пока не знаете что такое форма, то собственно обратитесь к либо к нашему, либо какому‑то другому источнику информации, посвященной html‑страницам. Форма — это один из стандартных элементов html. Так вот, собственно, на этой странице, которую мы пользователю уже должны отдать. Это может быть, например статическая страница, это может быть индекс html, которую мы ему, пользователю, возвращаем по первому заходу. И на ней он в форму заполняет нужные нам данные и отправляет. Что значит отправляет. Отправляет его в браузер, запрашивает наш сервер на определенном порту поднятый и мы от браузера получаем от пользователя request. И мы на этот request пишем свой сервлет. В этом сервлете (действия происходят на бекенде) получаем объект request. Нам с вами для того, чтобы, то есть мы из этого request сможем извлечь логин, email, пароль. Это будут строковые переменные. И после этого нам с вами нужно эти строковые переменные у себя запомнить. Для того, чтобы работать с этими данными не внутри сервлета, а в некой службе на сервере мы с вами создадим класс аккаунт сервис и у этого аккаунт сервиса будут специальные методы, которые предназначены для регистрации пользователя. То есть мы вызовем внутри сервлета аккаунт сервис, передадим ему параметры, которые получили из request. Внутри аккаунт сервиса мы с вами создадим еще один объект UserProfile. Этот UserProfile будет содержать поля логин, email и пароль. И каждый раз, когда пользователь захочет, каждый новый пользователь захочет зарегистрироваться, мы на него будем создавать UserProfile. И этот UserProfile пусть хранит в специальной карте. Мы создадим Map. Ключ в карте будет логин пользователя, строка. И ключ будет указывать на Value. Это UserProfile.

|  |
| --- |
| Авторизация  Frontend  1. Страница с вводом: логин, пароль  Backend  1. Сервлет обработки запроса  2. Класс AccountService с методом авторизации  3. Класс UserProfile в карте по логину  4. Карта HttpSession -> UserProfile |

Следующая страница, которая нам с вами понадобиться это страница авторизации. То есть когда пользователь уже прошел регистрацию и уже вызвал наш сервлет. В этом сервлете мы уже вызвали аккаунт сервис. В аккаунт сервис уже передали данные регистрации и запомнили там его. Запомнили, означает записали UserProfile в карту. Запись UserProfile, логин пользователя на UserProfile — это и есть собственно наш аналог записи в базу. То есть в будущем вы в этом месте мы не будем сохранять его в runtime, в карту логин на профиль, вы будем создавать запись в базе. Пока что у нас там заглушка рантаймовая и мы храним данные пользователя в аккаунт сервисе. Когда пользователь произвел эти действия он приходит к нам с авторизацией, и мы ему даем снова страницу. Опять же это может быть страница результата какого‑то сервлета, может быть просто статический файл. И на нем, на этой странице мы спрашиваем логин и пароль. То есть там тоже форма. На этой форме логин и пароль. Мы этот логин и пароль с пользователя спрашиваем. Он нам с request его запрашивает, отправляет. Мы на бекенде уже пишем еще один сервлет. Я предлагаю вам пока, чтобы не путаться, на каждый такой запрос заводить отдельный класс. Каждый раз, когда я говорю сервлет это означает что это наследник http сервлета, предназначен для работы с конкретным данным запросом. Сервлет обработки запросов в нем тоже будет ссылка на объект аккаунт сервиса, то есть у нас 2 сервлета, оба держат ссылку на один и тот же аккаунт сервис. И в аккаунт сервис мы еще добавим метод авторизации. Мы в этот метод авторизации положим логин и пароль, который достали из request пользователя и найдем в карте по логину UserProfil. То есть у нас смотрите, аккаунт сервис, в нем карта, в карте по ключу логина мы можем спросить UserProfil. Если мы его не нашли, значит профиль еще не зарегистрировался. Мы можем ему вернуть ответ зарегистрируйтесь сначала. Если пользователя мы нашли в этой карте, то мы приветствуем пользователя, говорим здравствуй пользователь и можем взять, например из этой карты еще его email. Написать здравствуй пользователь, мы знаем твой email. Мы поприветствовали его. Кроме того, нам хорошо здесь будет в аккаунт сервисе же хранить карту http сессии на UserProfile. Зачем это нужно, я расскажу четь позже.

|  |
| --- |
| Учетная запись  Frontend  1. Страница после логина  Backend  1. Сервлет обработки запроса страницы  2. Класс HttpSession из request  3. Класс UserProfile в карте по HttpSession |

И когда пользователь прошел авторизацию, мы можем показать ему страницу с его учетной записью. То есть это та страница, которую фронтенд покажет после логина пользователя. Jetty уже будет узнавать этого пользователя, она будет знать что это то же самый пользователь, потому что у него в куках будет прописан ключ для поиска http сессии. Мы разберем это позже в теме про http сессию. Пока что мы знаем, что есть такая страница с учетной записью, и на этой странице мы можем поприветствовать пользователя. То есть смотрите, когда пользователь обновляет эту страницу, запрос опять же приходит к нам на сервлет учетной записи, мы проверяем что пользователь залогинен. И мы можем получить http сессию из request. То есть в request Jetty нам заодно будет передавать не просто данные пользователя он нам отправит, но еще и некий объект, представляющий браузер пользователя. И мы сможем найти по http сессии юзерский профиль, то есть мы считаем, что пользователь уже залогинелся и мы его, у него есть соответсвия http сессии на UserProfile. То есть смотрите у зарегистрированных пользователей есть соответствие логин на UserProfile. У залогиненых пользователей есть соответствие http сессии, или http сессии toString дали нам ключ http сессии на юзерский профиль. И соответственно первой проверкой, проверкой если пользователь в карте логин на UserProfile мы узнаем зарегистрирован он или нет, то поиском по http сессии мы узнаем залогинен он или нет. И если мы уже знаем, что он залогинен, у нас есть объект профиля, в этом профиле мы уже сложили 3 поля, которые мы о нем уже знаем. Потом мы в него можем добавлять другую информацию о пользователе, например, результаты предыдущих взаимодействий с системы, время последнего захода, в общем все что хотите.

|  |
| --- |
| Выход  Frontend  1. Кнопка выхода  Backend  1. Сервлет обработки запроса  2. Класс HttpSession из request  3. Удаление UserProfile в карте по HttpSession |

После того, как пользователь вошел в систему, авторизовался, посмотрел свой профиль, как‑то по взаимодействовал с системой, мы должны дать возможность еще выйти из системы. И для того, чтобы пользователь имел возможность это сделать нам нужна кнопка выхода на фронтеде. Это тоже будет некая форма, которая отправит запрос на определенный сервлет. То есть нужно сделать на бекенде сервлет, который обработает выход пользователя. И если мы получили от пользователя запрос на выход, то мы вместе с этим запросом получили от него Http сессию. Как в случае, если он пользуется сервисом и ходит по странице. И когда мы по этой http сессии мы можем найти, у нас есть карта, в которой этот пользователь по http сессии может быть найден, может быть найден его профиль. Нам с вами просто нужно будет удалить из этой карты по сессии юзерский профиль, чтобы при следующем запросе мы увидели что пользователь не авторизован. Нам не нужно его удалять из записи логин на UserProfile, потому что в данном случае мы удалим запись о том, что он зарегистрировался. Нам нужно удалить запись о том, что он залогинен. Просто в карте сессия на профиль нужно будет по ключу этой карты запись удалить.

2.1.5 HttpSession

Давайте от технических деталей реализации коллекций, карт вернемся к нашей задаче. Теперь, когда вы знаете что такое коллекции, карты и что такое дженерики, вам будет проще понять как; вы можете сейчас вернуться к нашей задаче и еще раз посмотреть видео с задачей и уже лучше понимать какие именно, как именно нам нужно организовать аккаунт сервис наш, который будет у нас отвечать за хранение информации о регистрации, авторизации пользователей. И в нем, в этом аккаунт сервисе у нас с вами будет 2 карты. Первая карта будет логин пользователя на UserProfile. Вы уже знаете, что такое карта, что такое ключ и как по ключу происходит поиск элемента. И вторая карта, которая нам с вами понадобиться, это карта http сессия на UserProfile. Это карта, по которой мы будем мы узнавать авторизовался ли пользователь. Если первая нужна для того, чтобы понять зарегистрирован ли пользователь является нашей некой заглушкой для базы данных, то вторая уже вполне себе боевая сущность, которая у нас вполне останется прямо до конца работы, она хранит информацию о том, пользователь, который запрашивает у нас что­‑то, авторизован или нет. Запрашивает что‑то это означает, что обращается к одному из наших сервлетов и в этот сервлет передает свой запрос, то есть к нам сервлет придет request. И в request заодно есть еще такая сущность как http сессия. Это картинка к этому слайду она иллюстрирует работу с куками, то есть Jetty, когда работа с пользователем, она о себе информацию у пользователя в браузере записывает в куки. Когда пользователь запрашивает что‑то у веб сервера, в нашем случае запрос попадает в Jetty вместе с запросом пользователь заодно еще отправляет значение из куки и в куки, то есть по‑русски печенки. И в этой куки, в этой печенке записана информация http сессии, то есть Jetty сама по значению, которое присылает пользователь находит http сессию и нам в сервлет эту http сессию передает.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11. | HttpSession session = request.getSession();  Long userId = (Long) session.getAttribute(  'userId');  if (userId == null) {  userId = userIdGenerator  .getAndIncrement();  session.setAttribute('userId', userId);  }  String key = session.toString(); |

Вот собственно пример выглядит следующим образом. Вот этот код может быть может находиться внутри сервлета. Внутрь сервлета у нас попадает request. Если вы посмотрите на сигнатуру функцию doGet или doPost() внутри сервелета, то вы увидите, что попадает request и у этого request можно взять http сессию. Вот эта сессия. В сессию можно взять и сохранить что‑то свое, то есть вы можете спросить, а если в сессии, то есть в сессии внутри есть аналог карты, ассоциативный массив ключ‑значение. Вы можете спросит у нее атрибут. Если его нет, то вы можете этот атрибут задать. А если он есть, то вы можете взять и в качестве ключа в карте аккаунт сервиса, в которой мы пишем залогинен пользователь или нет, использовать, например, вот этот userId (Long). Либо если вы не хотите писать значения в сессию, вы можете взять у сессии строку, то есть превратить сессию toString(), вызвать у нее toString(), получить ключ сессии. Я не проверял, но скорей всего он будет совпадать со значением в куки. И у себя в карте хранить уже запись строка на userProfile. То есть в первом варианте у нас получится в аккаунт сервисе Map<Long, userPrifile>, а во втором случае у нас получится Map<String, userProfile>. Вариант какой выберете — на ваше усмотрение. Главное просто чтобы по запросу пользователя, по его request, можно было однозначно понять если для него профиль у нас в карте или нет для него профиля. Если нет, надо попросить его залогиниться.