1. **Для чего используется сервер приложений Tomcat?**
2. **Как установить Tomcat на операционную систему Windows?**
3. **Как проверить, что Tomcat успешно установлен и запущен на вашем компьютере?**

---

1. Для чего используется сервер приложений Tomcat?

- **Apache Tomcat** — это **сервлет-контейнер**, реализующий спецификации **Java Servlet, JSP и WebSocket**.
- Основные функции:
    1. Принимает HTTP-запросы и передаёт их веб-приложению.
    2. Управляет жизненным циклом сервлетов и JSP.
    3. Обеспечивает базовую инфраструктуру для Java web-приложений:
        - управление сессиями
        - обработка запросов
        - логирование
- В отличие от полноценных Java EE серверов (GlassFish, WildFly), Tomcat **не реализует EJB и JPA**, но может быть использован совместно с ними через Spring и другие фреймворки.

Итог: Tomcat — это лёгкий веб-сервер и контейнер сервлетов для запуска Java веб-приложений.

---

2. Как установить Tomcat на Windows

Шаги:

1. **Скачать Tomcat**
    - Перейти на официальный сайт: https://tomcat.apache.org
    - Выбрать версию (например, Tomcat 10.x)
    - Скачать **Core / Windows zip** или **Windows Service Installer**
2. **Распаковать Tomcat (если zip)**
    - Распаковать в удобную директорию, например `C:\Tomcat`
3. **Настроить переменные окружения**
    - Убедиться, что **Java JDK** установлена (`JAVA_HOME` → путь к JDK)
    - Добавить `JAVA_HOME/bin` в `PATH`, если необходимо
4. **Запуск Tomcat**
    - Если zip:
        - Перейти в папку `bin`
        - Запустить `startup.bat` (cmd от администратора не обязателен)
    - Если установщик: Tomcat создаёт **Windows Service**, можно запускать через Services или `Tomcat9.exe //IS//Tomcat9`

---

3. Как проверить, что Tomcat успешно установлен и запущен

Способы проверки:

1. **Через браузер**
   - Перейти на адрес: `http://localhost:8080`
   - Должна появиться стартовая страница Tomcat с надписью **"Apache Tomcat"**
2. **Через консоль**
   - После запуска `startup.bat` должны появиться сообщения вроде:

```
INFO: Server startup in 1234 ms
```

3. **Через Task Manager / Services**
   - Найти процесс `java.exe` или сервис `Tomcat`
   - Убедиться, что он запущен
4. **Через команду curl / PowerShell**

```
curl http://localhost:8080
```

- Если вернётся HTML-код стартовой страницы → сервер работает

Итог для собеседования:

Tomcat — это сервлет-контейнер для запуска Java веб-приложений. Устанавливается через скачивание zip или установщик, запуск через `startup.bat` или сервис, проверка через `http://localhost:8080` или вывод в консоли.

---

1. **Как сконфигурировать Tomcat для использования другого порта, отличного от стандартного 8080?**
2. **Как осуществляется деплой WAR-файла на сервер Tomcat?**
3. **Что такое файл server.xml в Tomcat и для чего он используется?**

---

1. Как сконфигурировать Tomcat для использования другого порта (не 8080)?

1. Перейти в папку Tomcat:
2. `TOMCAT_HOME/conf`
3. Открыть файл **`server.xml`** (об этом ниже).
4. Найти блок Connector:

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

4. Изменить порт на нужный, например:

```
<Connector port="9090" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

5. Сохранить файл и перезапустить Tomcat (`startup.bat` / сервис).

✅ После этого приложение будет доступно по новому порту:

`http://localhost:9090`

---

2. Как осуществляется деплой WAR-файла на сервер Tomcat?

Варианты деплоя:

1. **Через папку `webapps` (автоматический деплой)**

- Поместить `yourapp.war` в:

`TOMCAT_HOME/webapps/`

- Tomcat автоматически распакует WAR и создаст папку с приложением.
- Доступ через:

http://localhost:8080/yourapp/

2. **Через Tomcat Manager**

- Включить `manager` приложение (доступно по `http://localhost:8080/manager/html`)
- В разделе **Deploy** загрузить WAR-файл
- Tomcat развернёт приложение на сервере

3. **Ручной деплой через скрипт или CI/CD**

- Можно копировать WAR и через REST API Manager деплоить на удалённый сервер

---

3. Что такое файл `server.xml` в Tomcat и для чего он используется?

- **`server.xml`** — основной конфигурационный файл Tomcat, расположенный в:

```
TOMCAT_HOME/conf/server.xml
```

- Содержит ключевые настройки сервера:

Компонент	Назначение
`<Server>`	Определяет сервер и глобальные параметры (порт shutdown, слушатель)
`<Service>`	Группа соединителей и контейнеров приложений
`<Connector>`	Настройки портов, протоколов (HTTP/HTTPS)
`<Engine>`	Основной движок для обработки запросов, виртуальные хосты
`<Host>`	Виртуальные хосты, контексты, приложения
`<Context>`	Настройки конкретного приложения (пути, ресурсы, reloadable и т.д.)

Важно:

- Через `server.xml` можно менять порт, включать SSL, настраивать виртуальные хосты, управлять ресурсами.
- Не рекомендуется менять настройки, если можно обойтись `context.xml` для отдельного приложения.

---

1. **Как настроить Tomcat для работы с SSL/TLS?**
2. **Как реализовать кластеризацию в Tomcat для обеспечения высокой доступности приложений?**
3. **Как настроить Tomcat для использования внешнего источника данных (DataSource) для JNDI?**

---

1. Как настроить Tomcat для работы с SSL/TLS

1. **Создать или получить сертификат**
   - Через `keytool`:

```
keytool -genkey -alias tomcat -keyalg RSA -keystore keystore.jks -keysize
2048
```

- Ввести пароль и информацию о владельце.

2. **Скопировать keystore в Tomcat**
   Например: `TOMCAT_HOME/conf/keystore.jks`
3. **Настроить Connector в `server.xml`**

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
           maxThreads="200"
           SSLEnabled="true">
    <SSLHostConfig>
        <Certificate certificateKeystoreFile="conf/keystore.jks"
                     type="RSA"
                     certificateKeystorePassword="пароль"/>
    </SSLHostConfig>
</Connector>
```

4. **Перезапустить Tomcat**

- Доступ по HTTPS: `https://localhost:8443`

✅ Можно дополнительно включить:

- редирект с 8080 → 8443
- двухстороннюю аутентификацию (mutual TLS)

---

2. Как реализовать кластеризацию в Tomcat для высокой доступности приложений

Tomcat поддерживает **кластеры для сессий** и балансировку нагрузки.

Основные шаги:

1. **Включить кластер в `server.xml`**

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster">
    <Manager className="org.apache.catalina.ha.session.DeltaManager"
             expireSessionsOnShutdown="false"
             notifyListenersOnReplication="true"/>
    <Channel className="org.apache.catalina.tribes.group.GroupChannel">
        <Membership
className="org.apache.catalina.tribes.membership.McastService"
                    address="228.0.0.4" port="45564" frequency="500"
                    dropTime="3000"/>
        <Receiver
className="org.apache.catalina.tribes.transport.nio.NioReceiver"
                  address="auto" port="4000" autoBind="100"
                  selectorTimeout="100" maxThreads="6"/>
        <Sender
className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
            <Transport
className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
        </Sender>
    </Channel>
</Cluster>
```

2. **Обеспечить одинаковую конфигурацию приложений на всех нодах**

- Деплой WAR на каждую ноду

3. **Настроить балансировщик нагрузки (Apache HTTPD, Nginx, или hardware load balancer)**

- Раздаёт трафик между нодами

Эффект:

- Репликация сессий между нодами
- Высокая доступность приложения

---

3. Как настроить Tomcat для использования внешнего источника данных (DataSource) через JNDI

1. **Создать DataSource в `context.xml`** (или `server.xml`):

```
<Resource name="jdbc/MyDB"
          auth="Container"
          type="javax.sql.DataSource"
          maxTotal="20"
          maxIdle="10"
          maxWaitMillis="10000"
          username="dbuser"
          password="dbpass"
          driverClassName="org.postgresql.Driver"
          url="jdbc:postgresql://localhost:5432/mydb"/>
```

2. **Указать ресурс в web.xml (для JEE-style lookup)**

```
<resource-ref>
    <description>DB Connection</description>
    <res-ref-name>jdbc/MyDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

3. **Получить DataSource в коде через JNDI**

```
Context ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/MyDB");
Connection conn = ds.getConnection();
```

✅ Преимущества:

- Управление пулом соединений Tomcat
- Безопасное хранение паролей
- Снижение нагрузки на базу через connection pooling

---

1. **Что такое сервлет в контексте Java EE?**
2. **Как настроить Tomcat для запуска веб-приложения?**
3. **Каковы основные отличия между сервлетами и JSP?**

---

1. Что такое сервлет в контексте Java EE

- **Сервлет** — это **Java-класс**, который обрабатывает HTTP-запросы и формирует HTTP-ответы.
- Основные характеристики:
    1. Работает внутри **сервлет-контейнера** (например, Tomcat).
    2. Может обрабатывать GET, POST, PUT, DELETE и другие HTTP-методы.
    3. Управляется жизненным циклом контейнера (`init()`, `service()`, `destroy()`).

Пример простого сервлета:

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.getWriter().write("Hello, World!");
    }
}
```

- Доступ через: `http://localhost:8080/yourapp/hello`

---

2. Как настроить Tomcat для запуска веб-приложения

1) Через папку `webapps`

1. Собрать WAR-файл (`yourapp.war`).
2. Скопировать его в:

```
TOMCAT_HOME/webapps/
```

3. Tomcat автоматически распакует WAR и сделает доступным приложение:

```
http://localhost:8080/yourapp/
```

---

2) Через Tomcat Manager

1. Открыть `http://localhost:8080/manager/html`.
2. Авторизоваться (пользователь и пароль настраиваются в `tomcat-users.xml`).
3. Использовать форму **Deploy** для загрузки WAR.

---

3) Через конфигурацию Context (необязательно)

- Можно добавить Context в `conf/Catalina/localhost/yourapp.xml`:

```
<Context docBase="C:/apps/yourapp" path="/yourapp" reloadable="true"/>
```

- Позволяет запускать приложение из произвольной папки.

---

3. Основные отличия между сервлетами и JSP

Характеристика	Сервлет	JSP
Назначение	Логика обработки запросов	Представление (HTML с динамическим контентом)
Язык	Чистый Java	HTML + встроенные теги/скриптлеты (Java код)
Компиляция	Нет компиляции в WAR, запускается сразу	Сначала компилируется в сервлет контейнером, потом запускается как сервлет
Использование	Обработка данных, логика бизнес-уровня	Генерация HTML, вывод данных пользователю
Подход	Программирование через API	Более декларативный, для UI

Итог:

- JSP — это «удобный способ писать HTML с динамическим Java-кодом».
- Сервлет — это «Java-класс для обработки HTTP-запросов и бизнес-логики».
- Под капотом JSP компилируется в сервлет, поэтому по сути оба работают одинаково.

1. **Как передать данные из сервлета в JSP?**
2. **Как настроить контекстное меню веб-приложения в Tomcat?**
3. **Как использовать фильтры в сервлетах для управления доступом?**

---

1. Как передать данные из сервлета в JSP

В Java веб-приложениях **сервлет обрабатывает логику**, а JSP — отображает данные.

1) Через объект `request`

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        String message = "Hello from Servlet!";
        request.setAttribute("msg", message);  // передаем данные в JSP
        request.getRequestDispatcher("/hello.jsp").forward(request,
response);
    }
}
```

2) В JSP

```
<p>${msg}</p>
```

- Используется **EL (Expression Language)** для удобного доступа к атрибутам.
- `request.setAttribute()` → данные доступны **только на время текущего запроса**.

3) Через `session` или `application`

- `session.setAttribute("name", value)` — данные доступны для всех страниц сессии.
- `application.setAttribute("name", value)` — данные доступны глобально для всех пользователей.

---

2. Как настроить контекст веб-приложения в Tomcat

Контекст определяет «корневой путь» веб-приложения и его настройки.

1) Через папку `conf/Catalina/localhost`

- Создать файл `yourapp.xml`:

```
<Context docBase="C:/apps/yourapp" path="/yourapp" reloadable="true">
    <!-- Можно подключать ресурсы, фильтры, параметры -->
</Context>
```

- `docBase` — путь к WAR или распакованной папке приложения
- `path` — URL-префикс приложения (`http://localhost:8080/yourapp`)
- `reloadable="true"` — автоматическая перезагрузка при изменении классов

2) Через `META-INF/context.xml` в WAR

- Помещается прямо в WAR-файл
- Автоматически подключается при деплое

✅ **Преимущество:** можно управлять ресурсами и настройками приложения без изменения глобального `server.xml`.

---

3. Как использовать фильтры в сервлетах для управления доступом

Фильтр — это Java-класс, который **перехватывает запросы и ответы** до и после сервлета.

1) Создание фильтра

```
@WebFilter("/admin/*")
public class AuthFilter implements Filter {
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpSession session = req.getSession(false);

        if (session != null && session.getAttribute("user") != null) {
            chain.doFilter(request, response); // разрешаем доступ
        } else {
            ((HttpServletResponse) response).sendRedirect("/login.jsp");
        }
    }
}
```

2) Привязка фильтра

- Через аннотацию `@WebFilter("/path/*")`
- Или через `web.xml`:

```
<filter>
    <filter-name>AuthFilter</filter-name>
    <filter-class>com.example.AuthFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>AuthFilter</filter-name>
    <url-pattern>/admin/*</url-pattern>
</filter-mapping>
```

3) Зачем фильтры

- Аутентификация/авторизация
- Логирование и аудит
- Сжатие ответов
- Изменение заголовков или кодировки

---

1. **Как использовать паттерн MVC в разработке веб-приложения на сервлетах и JSP?**
2. **Как настроить подключение к базе данных через JNDI в Tomcat?**
3. **Какие существуют способы оптимизации производительности веб-приложения на сервлетах и JSP?**

---

1. Как использовать паттерн MVC в разработке веб-приложения на сервлетах и JSP

MVC (Model-View-Controller) — это архитектурный паттерн, разделяющий приложение на три слоя:

Слой	Описание	Пример в Java EE
Model (Модель)	Данные и бизнес-логика	POJO, DAO, JPA-сущности
View (Представление)	Отображение данных пользователю	JSP, HTML, JSTL, EL
Controller (Контроллер)	Обработка запросов и управление потоками	Сервлеты

Пример:

Сервлет (Controller):

```
@WebServlet("/users")
public class UserController extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        List<User> users = UserDAO.getAllUsers(); // Model
        request.setAttribute("users", users);     // Передача в View
        request.getRequestDispatcher("/users.jsp").forward(request,
response); // View
    }
}
```

JSP (View):

```
<ul>
<c:forEach var="user" items="${users}">
    <li>${user.name} (${user.email})</li>
</c:forEach>
</ul>
```

✅ **Итог:**

- Контроллер принимает HTTP-запрос → вызывает модель → передаёт данные в JSP → пользователь видит результат.

---

2. Как настроить подключение к базе данных через JNDI в Tomcat

1) Создать ресурс DataSource в Tomcat

`conf/context.xml` или `META-INF/context.xml` в приложении:

```
<Resource name="jdbc/MyDB"
          auth="Container"
          type="javax.sql.DataSource"
          username="dbuser"
          password="dbpass"
          driverClassName="org.postgresql.Driver"
          url="jdbc:postgresql://localhost:5432/mydb"
          maxTotal="20"
          maxIdle="10"
          maxWaitMillis="10000"/>
```

2) Объявить ресурс в `web.xml`

```
<resource-ref>
    <description>DB Connection</description>
    <res-ref-name>jdbc/MyDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

3) Получить DataSource через JNDI

```
Context ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/MyDB");
Connection conn = ds.getConnection();
```

✅ **Плюсы:**

- Управление пулом соединений Tomcat
- Безопасное хранение паролей
- Повышение производительности

---

3. Способы оптимизации производительности веб-приложения на сервлетах и JSP

Метод	Объяснение
Lazy initialization и минимизация SQL-запросов	Загружать только нужные данные, использовать batch-fetch, DTO
Кэширование данных	В памяти (EHCache, Redis) или на уровне сессии
Использование Connection Pool / JNDI DataSource	Избегать постоянного открытия/закрытия соединений с БД
Компиляция JSP в сервлеты заранее	Сокращает время на первый запрос
Минимизация объёмного HTML/JS/CSS	Сжатие, использование CDN, gzip
Фильтры для кэширования и сжатия	`GzipFilter`, `CachingFilter`
Оптимизация сессий	Минимизировать хранение больших объектов в сессии
Использование Content Delivery и асинхронной загрузки	AJAX, Servlet 3.0 async support

---