AXIOM JDK

ПРОГРАММНОЕ ИЗДЕЛИЕ
«СЕРВЕР ПРИЛОЖЕНИЙ LIBERCAT CERTIFIED»

Руководство пользователя

РОФ.06262534.00036-01 90-99 02

Листов 51

2024

АННОТАЦИЯ

Настоящий документ содержит сведения по эксплуатации изделия Программное обеспечение «Сервер приложений Libercat Certified» (далее Libercat Certified).

Содержание

1. Введение в Libercat Certified

Этот документ представляет собой краткое введение в понятия и терминологию изделия Сервер приложений Libercat Certified. Пользователям и веб-разработчикам необходимо ознакомиться с данным документом до начала работы.

1.1. Терминология

В документе присутствует ряд терминов. Некоторые из них специфичны для Libercat Certified, а другие определены [в спецификациях Servlet и JSP](https://wiki.apache.org/tomcat/Specifications).

- **Context** - В двух словах, контекст – это веб-приложение.

1.2. Каталоги и файлы

Вот некоторые из ключевых каталогов Libercat Certified:

- /bin - Запуск, выключение и другие сценарии. Файлы с расширением `*.sh` (для систем Unix) являются функциональными копиями файлов с расширением `*.bat` (для систем Windows). Так как командная строка Win32 не имеет достаточного функционала, здесь также присутствуют дополнительные файлы.
- /conf - Файлы конфигурации и связанные с ними DTD. Самый важный файл – это server.xml. Это главный файл конфигурации контейнера.
- /logs - По умолчанию здесь расположены файлы журнала.
- /webapps - Это место для установки веб-приложений.

1.3. CATALINA_HOME и CATALINA_BASE

В документации имеются ссылки на два следующих свойства:

CATALINA_HOME представляет собой корневую папку установки Libercat Certified, например:

```
/home/libercat/libercat-9.5.0
```

или

```
C:\Program Files\libercat-9.5.0
```

CATALINA_BASE представляет собой конфигурацию определенного экземпляра Libercat Certified. Если вы хотите иметь несколько экземпляров Libercat Certified на одной машине – используйте переменную CATALINA_BASE.

Если вы используете различные экземпляры, то CATALINA_HOME содержит статические данные, такие как файлы .jar или бинарные файлы. CATALINA_BASE содержит файлы конфигурации, журнала, развернутые приложения и другие необходимые ресурсы.

1.3.1. Для чего используется CATALINA_BASE

По умолчанию CATALINA_HOME и CATALINA_BASE указывают на один и тот же каталог. Отредактируйте CATALINA_BASE вручную, если вы хотите запускать несколько экземпляров Libercat Certified на одной машине. Благодаря этому вы получаете следующие преимущества:

- Более легкий переход на новую версию Libercat Certified. Так как все экземпляры с одним местоположением CATALINA_HOME используют один набор файлов .jar и двоичных файлов, вы можете с легкостью обновить файлы до новой версии и применять изменения ко всем экземплярам Libercat Certified, используя один каталог CATALINA_HOME.

- Отсутствие дублирования одних и тех же статических файлов .jar .
- Возможность совместного использования некоторых настроек, например, setenv или файла сценария bat (в зависимости от операционной системы).

1.3.2. Содержимое CATALINA_BASE

Перед началом использования CATALINA_BASE создайте дерево каталогов, используемое CATALINA_BASE. Примечание: если вы не создадите все рекомендуемые каталоги, Libercat Certified создаст их автоматически. Если автоматически создать каталог не удается, например из-за отсутствия разрешений, Libercat Certified либо не запустится, либо будет работать неправильно.

Рассмотрите следующий список каталогов:

- Каталог bin с файлами setenv.sh, setenv.bat и tomcat-juli.jar.

Рекомендуемый: Нет.
Порядок поиска: Сначала проверяется CATALINA_BASE; если не найден, то используется CATALINA_HOME.

- Каталог lib с ресурсами для добавления в classpath.

Рекомендуемый: Да, если ваше приложение зависит от внешних библиотек.
Порядок поиска: Сначала проверяется CATALINA_BASE; затем загружается CATALINA_HOME.

- Каталог logs для файлов журнала экземпляра.

Рекомендуемый: Да.

- Каталог webapps для автоматически загружаемых веб-приложений.

Рекомендуемый: Да, если вы хотите развертывать приложения.
Порядок поиска: Только CATALINA_BASE.

- Каталог work, содержащий временные рабочие каталоги для развернутых веб-приложений.

Рекомендуемый: Да.

- Каталог temp, используемый JVM для временных файлов.

Рекомендуемый: Да.

Рекомендуется не изменять файл tomcat-juli.jar. Однако, если вы хотите использовать собственную систему записи в журнал работы, вы можете заменить файл tomcat-juli.jar в CATALINA_BASE для определенного экземпляра Libercat Certified.

Также рекомендуем скопировать все файлы конфигурации из каталога CATALINA_HOME/conf в каталог CATALINA_BASE/conf. Если файлы конфигурации отсутствуют в CATALINA_BASE, они не будут прочитаны из CATALINA_HOME. В дальнейшем это может вызвать отказ в работе.

1.3.3. Как использовать CATALINA_BASE

CATALINA_BASE – это переменная окружения. Вы можете установить ее до запуска Libercat Certified, например:

- В Unix: CATALINA_BASE=/tmp/tomcat_base1 bin/catalina.sh start
- В Windows: CATALINA_BASE=C:\tomcat_base1 bin/catalina.bat start

См главу «Установка и настройка средства Libercat Certified» в *Руководстве администратора*.

2. Руководство по загрузчикам классов

2.1. Обзор

Как и многие серверные приложения, Libercat Certified устанавливает различные загрузчики классов (классы для реализации java.lang.ClassLoader) для того, чтобы предоставить порциям контейнера и веб-приложениям, запущенным в контейнере, доступ к различным репозиториям доступных классов и ресурсов. Этот механизм используется для реализации функциональности, определяемой в спецификации сервлета, версия 2.4 – в особенности, секции 9.4 и 9.6.

В среде Java загрузчики классов отображаются в виде дерева. Как правило, если загрузчик классов получает запрос на загрузку определенного класса или ресурса, он перенаправляет запрос сначала родительскому загрузчику классов, а затем выполняет поиск в своих репозиториях, только если родительский загрузчик классов не смог найти запрошенный класс или ресурс. Стоит помнить, что модель для загрузчиков классов веб-приложений немного отличается от этой, как будет рассмотрено позднее, но основные принципы не отличаются.

Когда Libercat Certified запускается, он создает набор загрузчиков классов, организованных в следующие родительски-дочерние отношения, где родительский загрузчик классов выше дочернего:

```
      Bootstrap
          |
       System
          |
       Common
       /     \
 Webapp1   Webapp2 ...
```

Характеристики каждого из этих загрузчиков классов, включая источник классов и ресурсов, которые они делают видимыми, подробно рассматриваются в следующем разделе.

2.2. Определения загрузчиков классов

Как показано на диаграмме выше, Libercat Certified создает следующие загрузчики классов во время инициализации:

- **Bootstrap** – Этот загрузчик классов содержит базовые классы среды выполнения, предоставляемой виртуальной машиной Java, а также другие классы из файлов JAR, находящихся в каталоге системных расширений ($JAVA_HOME/jre/lib/ext).

Примечание: некоторые виртуальные машины могут реализовать более одного загрузчика в данном случае, или он может быть невидим вовсе (как загрузчик классов).

- **System** – Этот загрузчик классов обычно инициализируется из содержимого переменной окружения CLASSPATH. Все такие классы видимы для внутренних классов Libercat Certified и веб-приложений. Однако стандартные скрипты запуска Libercat Certified ($CATALINA_HOME/bin/catalina.sh или %CATALINA_HOME%\bin\catalina.bat) полностью игнорируют содержимое переменной среды CLASSPATH, а вместо этого собирают системный загрузчик классов из следующих репозиториев:
  - $CATALINA_HOME/bin/bootstrap.jar – Содержит главный() метод, используемый для инициализации сервера Libercat Certified и реализации зависящего от него загрузчика классов.
  - $CATALINA_BASE/bin/tomcat-juli.jar или $CATALINA_HOME/bin/tomcat-juli.jar – Ведение журнала реализуемых классов. Они включают расширенные классы java.util.logging API, известные как Libercat Certified JULI, и копию переименованного пакета Apache Commons Logging library, используемую внутри Libercat Certified. См. logging documentation, чтобы увидеть подробности.

Если tomcat-juli.jar имеется в $CATALINA_BASE/bin, то используется вместо того, который находится в $CATALINA_HOME/bin. Это полезно для определенных конфигураций журнала.

- $CATALINA_HOME/bin/commons-daemon.jar – Классы из Apache Commons Daemon проекта. Этот файл JAR отсутствует в CLASSPATH собранном catalina.bat|.sh скриптами, но относится к файлу манифеста *bootstrap.jar*.
- **Common** – Этот загрузчик классов содержит дополнительные классы, которые видимы для внутренних классов Libercat Certified и всех веб-приложений.

Обычно классы приложений не должны размещаться здесь. Места поиска загрузчиком классов определяются свойством common.loader в $CATALINA_BASE/conf/catalina.properties. Настройка по умолчанию будет искать следующие местоположения в порядке их перечисления:

- Нераспакованные классы и ресурсы в $CATALINA_BASE/lib
- Файлы JAR в $CATALINA_BASE/lib
- Нераспакованные классы и ресурсы в $CATALINA_HOME/lib
- Файлы JAR в $CATALINA_HOME/lib

По умолчанию ищутся следующие файлы:

- *annotations-api.jar* – Классы аннотаций JavaEE.
- *catalina.jar* – Реализацию сервлета Catalina порции контейнера в Libercat Certified.
- *catalina-ant.jar* – Задания Libercat Certified Catalina Ant.
- *catalina-ha.jar* – Пакет высокой доступности.
- *catalina-ssi.jar* - Пакет включений на стороне сервера.
- *catalina-storeconfig.jar* – Генерация конфигурационных файлов XML из текущего состояния.
- *catalina-tribes.jar* – Пакет коммуникации групп.
- *ecj-*.jar* – Компилятор Eclipse JDT Java.
- *el-api.jar* – EL 3.0 API.
- *jasper.jar* – Компилятор и рабочая среда Libercat Certified Jasper JSP.
- *jasper-el.jar* – Реализация Libercat Certified Jasper EL.
- *jsp-api.jar* – JSP 2.3 API.
- *servlet-api.jar* – Сервлет 4.0 API.
- *tomcat-api.jar* – Некоторые интерфейсы, определяемые Libercat Certified.
- *tomcat-coyote.jar* – Коннекторы и классы утилит Libercat Certified.
- *tomcat-dbcp.jar* – Реализация пула подключения к базе данных, основанная на копии переименованного пакета Apache Commons Pool 2 и Apache Commons DBCP 2.
- *tomcat-i18n-**.jar* – Опциональные файлы JAR, содержащие ресурсы для других языков. Так как пакеты по умолчанию также включены в каждый из файлов JAR, их можно безопасно удалить, если нет необходимости вывода сообщений на других языках.
- *tomcat-jdbc.jar* – Альтернативная реализация пула подключения к базе данных, называемая Libercat Certified JDBC pool. См. ссылки на дополнительную документацию в Приложении 1.
- *tomcat-util.jar* – Общие классы, используемые различными компонентами Libercat Certified.
- *tomcat-websocket.jar* – Реализация WebSocket 1.1.
- *websocket-api.jar* – WebSocket 1.1 API.
- **WebappX** – Загрузчик классов создается для каждого веб-приложения, которое запускается как отдельный экземпляр Libercat Certified. Все несжатые классы и ресурсы в /WEB-INF/classes каталога веб-приложения, а также классы и ресурсы в файле JAR в /WEB-INF/lib каталоге веб-приложения делаются видимыми для этого веб-приложения, но не для других.

Как упомянуто выше, загрузчик классов веб-приложения расходится с моделью делегирования Java по умолчанию (в соответствии с рекомендациями в спецификации сервлетов, версия 2.4, раздел 9.7.2 Загрузчик классов веб-приложения). Когда загрузчиком классов обрабатывается запрос на загрузку класса из веб-приложения WebappX, данный загрузчик классов будет в первую очередь просматривать локальные репозитории вместо того, чтобы делегировать данную функцию Java до просмотра. Существуют исключения. Классы, являющиеся частью базовых классов JRE, не могут быть переопределены. Существуют исключения, такие как, например, компоненты анализатора XML, которые могут быть переопределены при помощи подходящей для этого функции виртуальной машины Java, а именно стандартной функции переопределения для Java версий 8 и ниже и функции обновления модулей для Java 9 и выше. Наконец, загрузчик классов веб-приложения всегда делегирует сначала классы JavaEE API для спецификаций, реализуемых Libercat Certified (Servlet, JSP, EL, WebSocket). Все другие загрузчики классов в Libercat Certified следуют обычному шаблону делегирования.

К тому же, с точки зрения веб-приложения, загрузка класса или ресурса из репозиториев происходит в следующем порядке:

- Загрузочные классы виртуальной машины Java.
- /WEB-INF/classes вашего веб-приложения.
- /WEB-INF/lib/*.jar вашего веб-приложения.
- Системный загрузчик классов (описан выше).
- Общий загрузчик классов (описан выше)

Если загрузчик классов веб-приложения настроен с опцией `<Loader delegate="true"/>`, то порядок становится следующим:

- Загрузочные классы виртуальной машины Java.
- Системный загрузчик классов (описан выше).
- Общий загрузчик классов (описан выше)
- /WEB-INF/classes вашего веб-приложения.
- /WEB-INF/lib/*.jar вашего веб-приложения.

2.3. Анализаторы XML и Java

Начиная с Java 1.4, копия JAXP API и анализатора XML упакована в JRE. Это влияет на приложения, которые использует собственный анализатор XML.

В старых версиях Libercat Certified можно было просто заменить анализатор XML в каталоге библиотек Libercat Certified и таким образом изменить анализатор для всех веб-приложений. Однако данный способ не будет эффективен, если вы пользуетесь современными версиями Java, потому что обычный процесс делегирования загрузчика классов всегда выберет реализацию внутри JDK вместо этого варианта.

Java <= 8 поддерживает механизм, называемый «Рекомендованные стандарты механизма переопределения» для реализации замещения API, созданного за пределами JCP (например, DOM и SAX из W3C). Он также может быть использован при обновлении реализации анализатора XML. Для получения подробной информации смотрите: http://docs.oracle.com/javase/1.5.0/docs/guide/standards/index.html. Для Java 9+, используйте функцию обновляемых модулей.

Libercat Certified использует рекомендованный механизм, включающий настройки системных свойств `-Djava.endorsed.dirs=$JAVA_ENDORSED_DIRS` в командной строке, запускающей контейнер. Значение опции по умолчанию: `$CATALINA_HOME/endorsed`. Этот рекомендованный каталог не создается по умолчанию. Помните, что рекомендованная функция больше не поддерживается в Java 9 и выше; указанное системное свойство будет задано только в случае, если существует каталог `$CATALINA_HOME/endorsed` или если переменная `JAVA_ENDORSED_DIRS` была установлена.

Помните, что замещение любого компонента JRE несет риск. Если замещающий компонент не предоставляет 100 % совместимую API (например, API Xerces не на 100 % совместима с XML API JRE), то существует риск, что Libercat Certified и/или запущенное приложение будет выдавать ошибки.

2.4. Выполнение под управлением диспетчера безопасности

В случае выполнения под управлением диспетчера безопасности, местоположения, из которых разрешено загружать классы, также будут зависеть от содержимого файла политики. См. главу «Конфигурирование Libercat Certified с помощью SecurityManager» в *Руководстве администратора* для получения более подробной информации.

2.5. Расширенная конфигурация

Более сложная иерархия загрузчиков классов также может быть сконфигурирована. См. диаграмму ниже. По умолчанию загрузчики классов Server и Shared не определены, и используется упрощенная иерархия, показанная выше. Данная более сложная иерархия может использоваться путем изменения значений `server.loader` и/или `shared.loader` свойств в `conf/catalina.properties`.

```
  Bootstrap
      |
    System
      |
    Common
     /  \
Server  Shared
         /  \
   Webapp1  Webapp2 ...
```

Загрузчик классов Server, видимый только для Libercat Certified и полностью невидимый для веб-приложений.

Загрузчика классов Shared, видимый для всех веб-приложений, и может быть использован для общего кода всех веб-приложений. Однако любые изменения в данном общем коде будут требовать перезапуска Libercat Certified.

3. Руководство по ядру Jasper 2 JSP

3.1. Введение

Libercat Certified 9.0 использует ядро Jasper 2 JSP для реализации спецификации [JavaServer Pages 2.3](https://wiki.apache.org/tomcat/Specifications).

Jasper 2 был переработан для значительного повышения производительности по сравнению с исходной версией. В дополнение к общему улучшению кода были внесены следующие изменения:

- JSP Custom Tag Pooling - Объекты Java, реализованные для настраиваемых тегов JSP, теперь можно хранить в пуле и использовать повторно. Это значительно повышает производительность страниц JSP с настраиваемыми тегами.
- Background JSP compilation - Если вы внесли изменения в откомпилированную страницу JSP, Jasper 2 может перекомпилировать ее в фоне. Скомпилированная до этого страница JSP будет по-прежнему доступна для обработки запросов. Когда новая страница будет успешно скомпилирована, она заменит старую. Это помогает повысить доступность страниц JSP на производственном сервере.
- Recompile JSP when included page changes - Jasper 2 теперь может определять, компилировалась ли страница со времени изменения JSP, и затем перекомпилировать родительскую JSP.
- JDT used to compile JSP pages - Компилятор Eclipse JDT Java теперь используется для компиляции исходного кода JSP java. Этот компилятор загружает зависимости исходного кода из контейнера загрузчика классов. Ant и javac по-прежнему можно использовать.

Jasper реализован при помощи класса сервлета `org.apache.jasper.servlet.JspServlet`.

3.2. Конфигурация

По умолчанию Jasper сконфигурирован для использования при разработке веб-приложений. Смотрите раздел [Промышленная конфигурация](#промышленная-конфигурация) для получения сведений по конфигурированию Jasper для использования на производственном сервере Libercat Certified.

Сервлет, реализующий Jasper, конфигурируется при помощи инициализирующих параметров глобального `$CATALINA_BASE/conf/web.xml`.

- **checkInterval** - Если разработка не удалась, а интервал проверки больше нуля, разрешена компиляция в фоне. Интервал проверки – это время в секундах между проверками на необходимость рекомпиляции страницы JSP и ее зависимых файлов. По умолчанию `0` секунд.
- **classdebuginfo** – Определяет нужно ли компилировать файл класса с информацией об отладке: `true` или `false`, по умолчанию `true`.

- **classpath** - Определяет путь класса для компиляции сгенерированных сервлетов. Этот параметр работает только, если атрибут ServletContext org.apache.jasper.Constants.SERVLET_CLASSPATH не установлен. Этот атрибут всегда устанавливается при использовании Jasper в Libercat Certified. По умолчанию путь класса создается динамически на основе текущего веб-приложения.
- **compiler** – Определяет какой компилятор использовать с Ant для компиляции страниц JSP. Подходящие значения для этого те же, что и для атрибута компиляции задания Ant https://ant.apache.org/manual/Tasks/javac.html#compilervalues . Если значение не задано, то компилятор Eclipse JDT Java будет использован вместо Ant по умолчанию. Здесь нет значения по умолчанию. Если этот атрибут задан, то `setenv.[sh|bat]` должен использоваться для добавления `ant.jar`, `ant-launcher.jar` и `tools.jar` в `CLASSPATH` переменной среды.
- **compilerSourceVM** – Определяет версию JDK с которой совместимы файлы источников (Значение по умолчанию: 1.8)
- **compilerTargetVM** - Определяет версию JDK с которой совместимы сгенерированные файлы (Значение по умолчанию: 1.8)
- **development** – Устанавливает использование Jasper в режиме разработчика. Если да, то частота проверки изменений JSP может быть задана путем изменения параметра TestInterval.true или false, по умолчанию true.
- **displaySourceFragment** – Устанавливает надо ли включать фрагмент источника в сообщения исключений. true или false, по умолчанию true.
- **dumpSmap** – Определяет должна ли информация SMAP для отладки JSR45 быть записана в файл. true или false, по умолчанию false. false если suppressSmap верно.
- **enablePooling** - Определяет, разрешен ли пулинг обработчика тегов. Это параметр компиляции. Он не изменит поведения JSP, которая уже была откомпилирована. true или false, по умолчанию true.
- **engineOptionsClass** - Позволяет задать параметры класса для настройки Jasper. Если отсутствует, то будет использован EmbeddedServletOptions по умолчанию. Этот параметр игнорируется, если запущен под SecurityManager.
- **errorOnUseBeanInvalidClassAttribute** – Устанавливает должен ли Jasper выдавать ошибку, если значение атрибута класса в процессе использования Bean не является правильным классом bean. true или false, по умолчанию true.
- **fork** – Определяет имеет ли Ant ответвление компиляции страницы JSP, которая происходит на отдельной от Libercat Certified JVM. true или false, по умолчанию true.
- **genStringAsCharArray** – Устанавливает должны ли текстовые строки генерироваться как массивы для повышения производительности в некоторых случаях. По умолчанию false.

- **ieClassId** - Значение class-id должно посылаться в Internet Explorer при использовании тегов jsp:plugin. По умолчанию clsid:8AD9C840-044E-11D1-B3E9-00805F499D93.
- **javaEncoding** - Кодирование файлов Java для генерации файлов источников. По умолчанию UTF8.
- **keepgenerated** – Определяет нужно ли хранить генерированный исходный код Java для каждой страницы вместо того, чтобы удалить его. true или false, по умолчанию true.
- **mappedfile** – Определяет нужно ли генерировать статический контент одним предложением печати на строку ввода, чтобы облегчить отладку. true или false, по умолчанию true.
- **maxLoadedJsps** - Максимальное число JSP, загружаемых для веб-приложения. Если загружено больше JSP, последние использованные JSP будут выгружены, поэтому количество загруженных JSP не превышает лимита за один раз. Значение ноль или меньше отражает отсутствие лимита. По умолчанию -1
- **jspIdleTimeout** - Время бездействия до выгрузки JSP в секундах. Если значение равно 0 или менее – не выгружается. По умолчанию -1
- **modificationTestInterval** - Приводит к тому, что JSP (и зависимые файлы) не проверяется на изменения на протяжении определенного временного интервала (в секундах) с того времени, как в последний раз прошла проверка JSP на изменения. Значение 0 означает, что JSP будет проверяться при каждом доступе. Используется только в режиме разработки. По умолчанию 4 секунд.
- **recompileOnFail** - Если компиляция JSP неудачна, нужно ли игнорировать modificationTestInterval и предпринимать попытку рекомпиляции во время следующего доступа. Используется только в режиме разработки и по умолчанию отключено, так как компиляция может быть дорогостоящей и расходовать излишнее количество ресурсов.
- **scratchdir** - Какой временный каталог использовать для компиляции страниц JSP? По умолчанию это рабочий каталог текущего веб-приложения. Этот параметр игнорируется, если запущен под SecurityManager.
- **suppressSmap** – Устанавливает нужно ли подавлять генерацию информации SMAP для отладки JSR45. true или false, по умолчанию false.
- **trimSpaces** - Следует ли удалять временный текст, содержащий почти одни пробелы из вывода (true), заменить ли его одним пробелом (single) или оставить без изменений (false). Расширенная опция удалит начальные и конечные пробелы из текста шаблона и заменит последовательности пробелов и символов перевода строки в тексте шаблона до одного перевода строки. Обратите внимание, что если страница JSP или тега задает параметр trimDirectiveWhitespaces = true, то это будет иметь приоритет. По умолчанию false.
- **xpoweredBy** - Определяет добавление генерируемым сервлетом ответного заголовка X-Powered-By. true или false, по умолчанию false.

- **strictQuoteEscaping** - При использовании выражений сценария для значений атрибута нужно ли применять правила избегания кавычек в JSP.1.6. true или false, по умолчанию true.
- **quoteAttributeEL** – Устанавливает нужно ли применять правила заключения атрибутов в кавычки, описанные в JSP.1.6, при использовании EL как значения атрибута на странице JSP. true или false, по умолчанию true.

Компилятор Java из Eclipse JDT включен как компилятор по умолчанию. Это продвинутый компилятор Java, загружающий все зависимости из загрузчика классов Libercat Certified, что очень помогает при компиляции больших объемов с сотнями JAR. На быстрых серверах это позволяет использовать повторные циклы рекомпиляции даже для больших страниц JSP.

Apache Ant, использовавшийся в предыдущих релизах Libercat Certified, можно использовать вместо нового компилятора, настроив атрибут компилятора, как объяснено выше.

Если вам нужно изменить настройки сервлета JSP для приложения, вы можете переопределить сервлет JSP в /WEB-INF/web.xml. Однако это может вызвать проблемы, если вы попытаетесь развернуть приложение в другом контейнере, поскольку класс сервлета JSP может быть не найден. Эту проблему можно обойти, используя специфичный для Libercat Certified дескриптор развертывания /WEB-INF/tomcat-web.xml. Формат идентичен /WEB-INF/web.xml. Он переопределит любые настройки по умолчанию, но не /WEB-INF/web.xml. Поскольку он специфичен для Libercat Certified, он будет обработан только при развертывании приложения на Libercat Certified.

3.3. Известные проблемы

Как описано в [bug 39089](https://bz.apache.org/bugzilla/show_bug.cgi?id=39089), известная проблема JVM, [bug 6294277](http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6294277), может вызвать ошибку `java.lang.InternalError: name is too long to represent` – при компиляции очень больших JSP. При возникновении данной проблемы ее можно устранить следующими способами:

- уменьшить размер JSP;
- отключить генерацию SMAP и поддержку JSR-045 настройкой `suppressSmap` к `true`.

3.4. Промышленная конфигурация

Главная оптимизация JSP – это предварительная компиляция JSP. Однако это может оказаться невозможным (например, когда используется функция

jsp-property-group) или невыполнимым, и в этом случае конфигурация сервлета Jasper становится критичной.

При использовании Jasper 2 на производственном сервере Libercat Certified вы должны сделать следующие изменения в конфигурации.

- development - Чтобы отключить компиляцию при проверке страниц JSP на доступ, установите `false`.
- genStringAsCharArray - Для генерации более эффективных массивов установите `true`.
- modificationTestInterval - Если нужна разработка, установите `true` по любой причине (такой как динамическая генерация JSP), установка большего значения этого параметра значительно повысит производительность.
- trimSpaces - Чтобы удалить ненужные байты из ответа, установите `true`.

3.5. Компиляция веб-приложения

Использование Ant – предпочтительный способ компиляции веб-приложений с использованием JSPC. Помните, что при предварительной компиляции JSP, информация SMAP будет включена только в финальные классы, если suppressSmap – ложь, а compile – истина. Используйте приведенный ниже сценарий (похожий сценарий включен в комплект «развертки»), чтобы предварительно скомпилировать веб-приложение:

```
<project name="Webapp Precompilation" default="all" basedir=".">

   <import file="${tomcat.home}/bin/catalina-tasks.xml"/>

   <target name="jspc">

    <jasper
              validateXml="false"
              uriroot="${webapp.path}"
              webXmlInclude="${webapp.path}/WEB-INF/generated_web.xml"
              outputDir="${webapp.path}/WEB-INF/src" />

  </target>

  <target name="compile">

    <mkdir dir="${webapp.path}/WEB-INF/classes"/>
    <mkdir dir="${webapp.path}/WEB-INF/lib"/>

    <javac destdir="${webapp.path}/WEB-INF/classes"
           debug="on" failonerror="false"
           srcdir="${webapp.path}/WEB-INF/src"
           excludes="**/*.smap">
      <classpath>
        <pathelement location="${webapp.path}/WEB-INF/classes"/>
        <fileset dir="${webapp.path}/WEB-INF/lib">
          <include name="*.jar"/>
        </fileset>
```

```
        <pathelement location="${tomcat.home}/lib"/>
        <fileset dir="${tomcat.home}/lib">
          <include name="*.jar"/>
        </fileset>
        <fileset dir="${tomcat.home}/bin">
          <include name="*.jar"/>
        </fileset>
      </classpath>
      <include name="**" />
      <exclude name="tags/**" />
    </javac>

  </target>

  <target name="all" depends="jspc,compile">
  </target>

  <target name="cleanup">
    <delete>
        <fileset dir="${webapp.path}/WEB-INF/src"/>
        <fileset
dir="${webapp.path}/WEB-INF/classes/org/apache/jsp"/>
    </delete>
  </target>

</project>
```

Можно использовать следующую команду для запуска сценария (заменяя токены базовыми путями Libercat Certified и путем к предварительно компилируемому веб-приложению):

```
$ANT_HOME/bin/ant -Dtomcat.home=<$TOMCAT_HOME>
-Dwebapp.path=<$WEBAPP_PATH>
```

Затем нужно добавить в дескриптор развертывания веб-приложения объявления и сопоставления с запрашиваемым URL для сервлетов, которые были сгенерированы во время прекомпиляции. Вставьте `${webapp.path}/WEB-INF/generated_web.xml` в нужное место внутри файла `${webapp.path}/WEB-INF/web.xml`. Перезапустите веб-приложение (при помощи диспетчера) и протестируйте его на надежную работу с прекомпилированными сервлетами. Для автоматической вставки объявлений и сопоставлений сервлета с запрашиваемым URL с помощью фильтрации Ant может быть использован подходящий для этого токен, помещенный в дескриптор развертывания веб-приложения. Вот так обычно происходит автоматическая компиляция веб-приложения в Libercat Certified как часть процесса сборки.

В задаче Jasper вы можете использовать параметр `addWebXmlMappings` для автоматического соединения `${webapp.path}/WEB-INF/generated_web.xml` с текущим дескриптором развертывания веб-приложения `${webapp.path}/WEB-INF/web.xml`.

Если вы хотите использовать определенную версию Java для JSP, добавьте атрибуты компилятора javac source и target с соответствующими значениями. Например, 16 для компиляции JSP для Java 16.

Для работы на продакшен системе вы можете отключить отладочную информацию с помощью debug = "off".

Если вы не хотите останавливать генерацию JSP при первой ошибке синтаксиса JSP, используйте `failOnError="false"`, а с `showSuccess="true"` вся успешная JSP to Java генерация выводится на печать. Иногда очень помогает, если вы чистите и генерируете файлы источников java на `${webapp.path}/WEB-INF/src` и компилируете классы сервлета JSP на `${webapp.path}/WEB-INF/classes/org/apache/jsp`.

Подсказки:

- Когда вы переходите на другой релиз Libercat Certified, сгенерируйте заново и рекомпилируйте JSP с новой версией Libercat Certified.
- Используйте свойства системы java в среде сервера, чтобы запретить пулинг PageContext `org.apache.jasper.runtime.JspFactoryImpl.USE_POOL=false` и лимитировать буфер с `org.apache.jasper.runtime.BodyContentImpl.LIMIT_BUFFER=true`. Учитывайте, что изменение значений по умолчанию может влиять на производительность, но изменяет зависимость приложения.

3.6. Оптимизация

Существует некоторое количество расширений в Jasper, позволяющих пользователю оптимизировать поведение среды.

Первое из них – механизм присоединения тегов. Он позволяет использовать альтернативную реализацию управления тегами в веб-приложении. Плагины тегов регистрируются через `tagPlugins.xml` файл, который находится под `WEB-INF`. Плагин-пример для JSTL поставляется с Jasper.

Следующее расширение – интерпретатор языка выражений. Альтернативные интерпретаторы можно настроить через `ServletContext`. Более подробно см. страницу E`LInterpreterFactory` javadoc для дополнительной информации по конфигурации альтернативного интерпретатора EL. Альтернативный интерпретатор, в первую очередь нацеленный на настройку тегов, находится в `org.apache.jasper.optimizations.ELInterpreterTagSetters`. См. Javadoc для получения подробной информации об оптимизациях и их влиянии на соответствие спецификации.

Также предоставляется точка расширения для приведения строковых значений к Enum: `org.apache.jasper.optimizations.StringInterpreterEnum`. См. Javadoc для получения подробной информации об оптимизациях и их влиянии на соответствие спецификации.

4. Характеристика сервлета по умолчанию

4.1. Описание

Сервлет по умолчанию – это сервлет, который обслуживает статические ресурсы и перечни файлов каталогов (если перечни файлов каталогов активны).

4.2. Место декларирования

Сервлет по умолчанию глобально декларируется в $CATALINA_BASE/conf/web.xml. Объявление по умолчанию:

```
<servlet>
    <servlet-name>default</servlet-name>
    <servlet-class>
      org.apache.catalina.servlets.DefaultServlet
    </servlet-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>0</param-value>
    </init-param>
    <init-param>
        <param-name>listings</param-name>
        <param-value>false</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

...

<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

Сервлет по умолчанию загружается одновременно с запуском веб-приложения, а перечни файлов каталогов и отладка отключены.

Если вам нужно изменить настройки `DefaultServlet` для приложения, вы можете переопределить DefaultServlet в /WEB-INF/web.xml. Однако это вызовет проблемы, если вы попытаетесь развернуть приложение в другом контейнере, поскольку класс `DefaultServlet` не будет распознан. Эту проблему можно обойти, используя специфичный для Libercat Certified дескриптор развертывания `/WEB-INF/tomcat-web.xml`. Его формат идентичен формату `/WEB-INF/web.xml`. Он переопределит любые настройки по умолчанию, но не настройки в /WEB-INF/web.xml. Поскольку он специфичен для Libercat Certified, он будет обработан только при развертывании приложения в Libercat Certified.

4.3. Параметры

Сервлет по умолчанию предоставляет следующие параметры:

Свойство	Описание
debug	Уровень отладки. Это полезно только для разработчиков Libercat Certified. Как описано, полезные значения 0, 1, 11. [0]
listings	Если приветственный файл отсутствует, может ли отображаться перечень файлов каталога. значение может быть **true** или **false** [false]. Приветственные файлы – это часть api сервлета. **ВНИМАНИЕ**: Списки каталогов, состоящие из множества записей, довольно объемны. Множественные запросы к обширным перечням файлов каталогов могут поглощать значительные объемы ресурсов сервера.
precompressed	Если есть сжатая версия файла (файла с расширением .br или .gz добавленному к имени файла, расположенному вместе с оригинальным файлом), Libercat Certified обработает сжатый файл, если агент пользователя поддерживает соотнесение содержимого (br или zip) и активна данная опция. [false] Сжатый файл с .br или .gz расширением будет доступен в случае прямого запроса, если ресурс защищен политикой безопасности, сжатые версии должны быть также защищены. Также можно настраивать список форматов сжатия. Синтаксис – список пар, разделенный запятой. [content-encoding]=[file-extension] Например: br=.br,gzip=.gz,bzip2=.bz2. Если указано несколько форматов, клиент поддерживает более одного и не выражено предпочтение, порядок форматов в списке будет рассматриваться в том же порядке, как и на сервере, и использоваться для выбора возвращаемого формата.
readmeFile	Если представлен перечень файлов каталога, файл сведений тоже может быть представлен в виде списка. Этот файл вставляется «как есть», поэтому может содержать HTML.
globalXsltFile	Если вы желаете настроить перечень файлов каталогов, вы можете использовать трансформацию XSL. Значение – имя файла (в $CATALINA_BASE/conf/ или $CATALINA_HOME/conf/), который будет использован для всех перечней файлов каталогов. Его можно переопределить по содержимому и/или каталогу. См. **contextXsltFile** и **localXsltFile** ниже. Формат xml указан ниже.
contextXsltFile	Вы также можете настроить перечень файлов каталогов по содержимому, настроив contextXsltFile. Здесь должен быть путь относительно контекста (например: /path/to/context.xslt) к файлу с .xsl или .xslt расширением. Эта операция переопределяет globalXsltFile. Если это значение присутствует, но файл не существует, то globalXsltFile будет использовано. Если globalXsltFile не существует, то будет отображен перечень файлов каталогов по умолчанию.

Свойство	Описание
localXsltFile	Вы также можете настроить перечень файлов каталогов по каталогу, настроив localXsltFile. Это должен быть файл в каталоге, где будет находиться список, с .xsl или .xslt расширением. Эта операция переопределяет globalXsltFile и contextXsltFile. Если это значение присутствует, но файл не существует, то contextXsltFile будет использовано. Если contextXsltFile не существует, то globalXsltFile будет использовано. Если globalXsltFile не существует, то будет отображен перечень файлов каталогов по умолчанию.
input	Размер входного буфера (в байтах) для обработки считываемых ресурсов. [2048]
output	Размер выходного буфера (в байтах) при обработке записываемых ресурсов. [2048]
readonly	Это содержимое «только для чтения», то есть команды HTTP, такие как PUT и DELETE, запрещены? [true]
fileEncoding	Используемая кодировка файла при чтении статических ресурсов. [по умолчанию]
useBomIfPresent	Если статический файл содержит метку порядка байтов (BOM), следует ли использовать ее для определения кодировки файла, а не fileEncoding. Этот параметр должен иметь одно из значений true (удалить BON и использовать ее вместо fileEncoding), false (удалить BOM, но не использовать ее) или pass-through (не использовать BOM и не удалять ее). [true]
sendfileSize	Если используемый коннектор поддерживает sendfile, это означает, что минимальный размер файла в Кб, для которого будет использоваться sendfile. Используйте отрицательное значение, чтобы выключить sendfile. [48]
useAcceptRanges	Если значение верно, заголовок Accept-Ranges будет задан для ответа там, где необходимо. [true]
showServerInfo	Должна ли информация сервера быть представлена в ответе, отправляемом клиентам, если перечень файлов каталогов активен. [true]
sortListings	Должен ли сервер сортировать списки в каталоге. [false]
sortDirectoriesFirst	Должен ли сервер отображать все каталоги перед всеми файлами. [false]
allowPartialPut	Должен ли сервер обрабатывать HTTP-запрос PUT с заголовком Range как частичный PUT? Обратите внимание, что RFC 7233 поясняет, что заголовки RANGE действительны только для запросов GET. [true]

4.4. Настройка перечня файлов каталогов

Вы можете переопределить сервлет по умолчанию вашей собственной реализацией и использовать ее в объявлении web.xml. Если вы можете понять только что сказанное, мы предполагаем, что вы можете прочесть код сервлета по умолчанию и сделать нужные правки. (Если нет, то этот способ не для вас)

Вы можете использовать `localXsltFile`, `contextXsltFile` или `globalXsltFile`, и сервлет по умолчанию создаст документ xml и запустит его в процессе трансформации xsl на основе предоставленных значений. Сначала проверяется `localXsltFile`, затем `contextXsltFile`, и, наконец `globalXsltFile`. Если файлы XSLT не настроены, используется поведение по умолчанию.

Формат:

```
<listing>
  <entries>
    <entry type='file|dir' urlPath='aPath' size='###' date='gmt
date'>
      fileName1
    </entry>
    <entry type='file|dir' urlPath='aPath' size='###' date='gmt
date'>
       fileName2
    </entry>
    ...
    </entries>
  <readme></readme>
</listing>
```

- размер будет отсутствовать, если `type='dir'`
- Файл сведений – это ввод CDATA

Ниже пример xsl-файла, отображающий поведение Libercat Certified по умолчанию:

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="3.0">

  <xsl:output method="html" html-version="5.0"
    encoding="UTF-8" indent="no"
    doctype-system="about:legacy-compat"/>

  <xsl:template match="listing">
   <html>
    <head>
      <title>
        Sample Directory Listing For
        <xsl:value-of select="@directory"/>
      </title>
      <style>
        h1 {color : white;background-color : #0086b2;}
        h3 {color : white;background-color : #0086b2;}
        body {font-family : sans-serif,Arial,Tahoma;
```

```
            color : black;background-color : white;}
        b {color : white;background-color : #0086b2;}
        a {color : black;} HR{color : #0086b2;}
        table td { padding: 5px; }
      </style>
    </head>
    <body>
      <h1>Sample Directory Listing For
           <xsl:value-of select="@directory"/>
      </h1>
      <hr style="height: 1px;" />
      <table style="width: 100%;">
        <tr>
          <th style="text-align: left;">Filename</th>
          <th style="text-align: center;">Size</th>
          <th style="text-align: right;">Last Modified</th>
        </tr>
        <xsl:apply-templates select="entries"/>
        </table>
      <xsl:apply-templates select="readme"/>
      <hr style="height: 1px;" />
      <h3>Libercat/9.0</h3>
    </body>
   </html>
  </xsl:template>

  <xsl:template match="entries">
    <xsl:apply-templates select="entry"/>
  </xsl:template>

  <xsl:template match="readme">
    <hr style="height: 1px;" />
    <pre><xsl:apply-templates/></pre>
  </xsl:template>

  <xsl:template match="entry">
    <tr>
      <td style="text-align: left;">
        <xsl:variable name="urlPath" select="@urlPath"/>
        <a href="{$urlPath}">
          <pre><xsl:apply-templates/></pre>
        </a>
      </td>
      <td style="text-align: right;">
        <pre><xsl:value-of select="@size"/></pre>
      </td>
      <td style="text-align: right;">
        <pre><xsl:value-of select="@date"/></pre>
      </td>
    </tr>
  </xsl:template>

</xsl:stylesheet>
```

4.5. Как обезопасить перечень файлов каталогов

Используйте web.xml в каждом отдельном веб-приложении. См. раздел по безопасности в спецификации сервлета.

5. Руководство по коннекторам

5.1. Введение

Выбор коннектора для использования с Libercat Certified может быть трудным. На данной странице будут перечислены коннекторы, которые поддерживает данный релиз Libercat Certified, чтобы помочь вам сделать правильный выбор в соответствии с вашими потребностями.

5.2. HTTP

Коннектор HTTP настраивается по умолчанию с Libercat Certified и готов к использованию. Этот коннектор имеет минимальное время задержки и максимальную общую производительность.

Для кластеризации подсистема балансировки нагрузки HTTP с поддержкой «липкости» веб-сессий должна быть установлена для направления трафика на серверы Libercat Certified. Libercat Certified поддерживает mod_proxy (на Apache HTTP Server 2.x, и включен по умолчанию в Apache HTTP Server 2.2) в качестве подсистемы балансировки нагрузки. Необходимо отметить, что производительность проксирования HTTP обычно ниже, чем производительность AJP, поэтому предпочитаемой часто является кластеризация AJP.

5.3. AJP

При использовании одного сервера производительность с использованием нативного веб-сервера перед экземпляром Libercat Certified большую часть времени существенно ниже, чем отдельный Libercat Certified со своим HTTP-коннектором по умолчанию, даже если большая часть веб-приложения составлена из статичных файлов. Если интеграция с нативным веб-сервером требуется по любой причине, AJP-коннектор будет обеспечивать более высокий темп работы, чем проксированный HTTP. AJP-кластеризация наиболее эффективна с точки зрения Libercat Certified. По всем остальным параметрам она функционально эквивалентна кластеризации HTTP.

Нативные коннекторы, поддерживаемые данным релизом Libercat Certified, следующие:

- JK 1.2.x с любым из поддерживаемых серверов

- mod_proxy на Apache HTTP Server 2.x (включенном по умолчанию в Apache HTTP Server 2.2) с активированным AJP

Другие нативные коннекторы, поддерживающие AJP, могут работать, но больше не поддерживаются.

6. Ведение журнала в Libercat Certified

6.1. Введение

Внутреннее ведение журнала для Libercat Certified использует JULI, упакованную переименованную ветвь [Apache Commons Logging](https://commons.apache.org/logging) , которая жестко закодирована для использования платформы `java.util.logging`. Это гарантирует, что внутреннее ведение журнала контейнером Libercat Certified и ведение журнала любым веб-приложением останутся независимыми, даже если веб-приложение использует Apache Commons Logging.

Чтобы настроить Libercat Certified на использование альтернативной платформы ведения журнала, следуйте инструкциям, предоставленным альтернативной платформой ведения журнала для перенаправления ведения журнала для приложений, использующих `java.util.logging`. Имейте в виду, что нужно будет, чтобы альтернативная платформа ведения журнала была способна работать в среде, где в разных загрузчиках классов могут существовать разные средства ведения журнала с одним и тем же именем.

Веб-приложение, работающее с Libercat Certified, может:

- Использовать любую платформу ведения журнала по своему выбору.
- Использовать API системного ведения журнала, `java.util.logging`.
- Использовать API ведения журнала, предоставленный спецификацией Java Servlets, `javax.servlet.ServletContext.log(...)`

Платформы ведения журнала, используемые разными веб-приложениями, являются независимыми. См. главу “Руководство по загрузчикам классов”, чтобы увидеть подробности. Исключением из этого правила является `java.util.logging`. Если оно используется прямо или косвенно вашей библиотекой ведения журнала, тогда его элементы будут использоваться веб-приложениями совместно, потому что оно загружается загрузчиком системного класса.

6.1.1. Java API для ведения журнала — java.util.logging

Libercat Certified имеет собственную реализацию нескольких ключевых элементов `java.util.logging` API. Эта реализация называется JULI. Ключевым элементом здесь является настраиваемая реализация LogManager, которая осведомлена о разных веб-приложениях, работающих с Libercat Certified (и их разных загрузчиках классов). Она поддерживает частные конфигурации ведения журналов для каждого приложения. Она также получает уведомление от Libercat Certified, когда веб-приложение выгружается из памяти, также ссылки на его классы могут быть очищены, предотвращая утечки памяти.

Эта реализация `java.util.logging` активируется путем предоставления определенных системных свойств при запуске Java. Скрипты запуска Libercat Certified делают это за вас, но, если вы используете другие инструменты запуска Libercat Certified (такие, как jsvc, или запускаете Libercat Certified изнутри IDE), вам следует позаботиться о них самостоятельно.

Более подробные сведения о java.util.logging можно найти в документации по вашему JDK и на страницах Javadoc для пакета `java.util.logging`.

Более подробные сведения о Libercat Certified JULI можно найти ниже.

6.1.2. API ведения журнала сервлетов

Вызовы `javax.servlet.ServletContext.log(...)` для записи сообщений журнала обрабатываются внутренним ведением журнала Libercat Certified. Такие сообщения регистрируются в категории под следующим именем:

```
org.apache.catalina.core.ContainerBase.[${engine}].[${host}].[${conte
xt}]
```

Это ведение журнала выполняется в соответствии с конфигурацией ведения журнала Libercat Certified. Вы не можете перезаписать ее в веб-приложение.

API ведения журнала сервлетов предшествует по времени `java.util.logging` API, который сейчас предоставляется Java. Поэтому он не предлагает вам много опций. Например, вы не можете управлять уровнями ведения журнала. Тем не менее можно отметить, что в реализации Libercat Certified вызовы `ServletContext.log(String)` или `GenericServlet.log(String)` регистрируются на уровне INFO. Вызовы `ServletContext.log(String, Throwable)` или `GenericServlet.log(String, Throwable)` регистрируются на уровне SEVERE.

6.1.3. Консоль

При запуске Libercat Certified в операционных системах unix вывод с консоли обычно перенаправляется в файл под именем `catalina.out`. Это имя можно конфигурировать с помощью переменной среды. (См. скрипты запуска.) Все, что

записывается в `System.err/out`, будет зафиксировано в этом файле. Сюда может входить следующее:

- Незафиксированные исключения, распечатываемые функцией `java.lang.ThreadGroup.uncaughtException(..)`
- Дампы потоков, если вы затребовали их через системный сигнал

В случае запуска в виде службы в Windows вывод с консоли также фиксируется и перенаправляется, но имена файлов другие.

Принимаемая по умолчанию в Libercat Certified конфигурация ведения журнала записывает одни и те же сообщения в консоль и в файл журнала. Это великолепно при использовании Libercat Certified для разработки, но обычно не нужно при промышленной эксплуатации.

Старые приложения, которые продолжают использовать `System.out` или `System.err`, можно обмануть, установив атрибут `swallowOutput` в Context. Если этот атрибут установлен в значение `true`, вызовы `System.out/err` во время обработки запросов будут перехватываться, а их результат будет подаваться в подсистему ведения журнала с помощью вызовов `javax.servlet.ServletContext.log(...)`.

ПРИМЕЧАНИЕ: Функция `swallowOutput` фактически является трюком, и она имеет свои ограничения. Она работает только с вызовами `System.out/err` и только во время цикла обработки запроса. Она может не работать в других потоках, которые могут создаваться приложением. Ее нельзя использовать для перехвата платформ ведения журналов, которые сами выполняют запись в системные потоки, поскольку эти платформы запускаются заранее и могут получить прямую ссылку на потоки, прежде чем произойдет перенаправление.

6.1.4. Ведение журнала доступа

Ведение журнала доступа является связанной, но другой функцией, которая реализуется как вентиль. Она использует независимую логику для записи своих файлов журнала. Важнейшим требованием для ведения журнала доступа является обработка большого непрерывного потока данных с небольшим количеством служебных данных, поэтому она использует только Apache Commons Logging для собственных отладочных сообщений. Такой подход к реализации позволяет избежать дополнительных служебных данных и потенциально сложной конфигурации. Обратитесь к документации по Valves для получения более подробных сведений по ее конфигурации, включая различные форматы отчетов (см. Приложение 1).

6.2. Использование java.util.logging (по умолчанию)

Принимаемая по умолчанию реализация java.util.logging, предусмотренная в JDK, слишком ограниченная, чтобы быть полезной. Ключевым ограничением является

невозможность получить ведение журнала для каждого веб-приложения, поскольку конфигурация предусматривает работу с каждой VM. В результате Libercat Certified будет (в конфигурации по умолчанию) заменять принимаемую по умолчанию реализацию LogManager на дружественную по отношению к контейнеру реализацию под названием JULI, в которой отсутствуют эти недостатки.

JULI поддерживает те же механизмы конфигурации, что и стандартный JDK `java.util.logging`, используя либо программный подход, либо файлы свойств. Основная разница заключается в том, что могут быть заданы файлы свойств для каждого загрузчика классов (что позволяет легко заново развернуть дружественную webapp-конфигурацию), а файлы свойств поддерживают расширенные конструкции, дающие больше свободы для определения обработчиков и их назначения средствам ведения журнала.

JULI активируется по умолчанию и поддерживает конфигурацию для каждого загрузчика классов, вдобавок к обычной глобальной конфигурации java.util.logging. Это означает, что ведение журнала может быть сконфигурировано на следующих уровнях:

- Глобально. Это обычно выполняется в файле `${catalina.base}/conf/logging.properties`. Этот файл определяется системным свойством `java.util.logging.config.file`, которое задается скриптами запуска. Если он не читается или не сконфигурирован, вариантом по умолчанию является использование файла `${java.home}/lib/logging.properties` в JRE.
- В веб-приложении. Файл `WEB-INF/classes/logging.properties`

По умолчанию `logging.properties` в JRE определяет `ConsoleHandler`, который направляет ведение журнала в System.err. По умолчанию `conf/logging.properties` в Libercat Certified также добавляет несколько обработчиков `AsyncFileHandler`, которые выполняют запись в файлы.

По умолчанию пороговым значением уровня ведения журнала обработчика является `INFO`, которое можно изменить на `SEVERE`, `WARNING`, `CONFIG`, `FINE`, `FINER`, `FINEST` или `ALL`. Вы можете также задать определенные пакеты, из которых нужно собирать данные для ведения журнала, и указать уровень.

Чтобы разрешить ведение журнала отладки для части внутренних компонентов Libercat Certified, вам нужно настроить как соответствующие средства ведения журнала, так и соответствующие обработчики на использование уровня `FINEST` или `ALL`, например:

```
org.apache.catalina.session.level=ALL
java.util.logging.ConsoleHandler.level=ALL
```

При задействовании ведения журнала отладки рекомендуется, чтобы оно было активировано для области, самой узкой из возможных, поскольку ведение журнала отладки может генерировать большие объемы информации.

Используемая JULI конфигурация точно такая же, как и та, что поддерживается простым `java.util.logging`, но она использует несколько расширений для обеспечения большей гибкости при конфигурировании средств ведения журнала и обработчиков. Основные отличия:

- К именам обработчиков могут добавляться префиксы, чтобы можно было создавать экземпляры нескольких обработчиков одного класса. Префикс является строкой, которая начинается с цифры и заканчивается точкой «`.`». Например, `22foobar.` является действительным префиксом.
- Замена системного свойства выполняется для значений свойств, которые содержат `${systemPropertyName}`.
- Если используется загрузчик класса, который реализует интерфейс org.apache.juli.WebappProperties (загрузчик класса веб-приложения контейнера Libercat Certified так и делает), тогда замена свойств выполняется также для `${classloader.webappName}`, `${classloader.hostName}` и `${classloader.serviceName}`, которые заменяются именем веб-приложения, именем хоста и именем службы соответственно.
- По умолчанию средства ведения журнала не будут выполнять делегирование их родительскому элементу, если у них есть ассоциированные обработчики. Это можно изменить для каждого отдельного средства ведения журнала с помощью свойства `loggerName.useParentHandlers`, которое принимает булево значение.
- Корневое средство ведения журнала может определить свой набор обработчиков с помощью свойства `.handlers`.
- По умолчанию файлы журнала будут храниться в файловой системе 90 дней. Это можно изменить для каждого отдельного обработчика с помощью свойства `handlerName.maxDays`. Если для свойства указано значение `≤0`, тогда файлы журнала будут храниться в файловой системе вечно, в ином случае они будут храниться в течение указанного максимального количества дней.

Имеется несколько дополнительных классов реализации, которые можно использовать вместе с теми, что предоставляются Java. Наиболее известными являются `org.apache.juli.FileHandler` и `org.apache.juli.AsyncFileHandler`.

`org.apache.juli.FileHandler` поддерживает буферизацию журналов. Буферизация не включена по умолчанию. Чтобы сконфигурировать ее, используйте свойство `bufferSize` обработчика. Значение `0` использует буферизацию по умолчанию (обычно будет использоваться буфер объемом 8K). Значение `<0` принуждает средство записи сбрасывать данные после каждой записи в журнал. Значение `>0` использует BufferedOutputStream с заданным значением, но имейте в виду, что системная буферизация по умолчанию также применяется.

`org.apache.juli.AsyncFileHandler` является подклассом `FileHandler`, который ставит сообщения журнала в очередь и асинхронно записывает их в файлы журнала.

Его дополнительное поведение можно настроить, установив некоторые свойства в system properties.

Пример файла logging.properties, помещаемого в $CATALINA_BASE/conf:

```
handlers = 1catalina.org.apache.juli.FileHandler, \
           2localhost.org.apache.juli.FileHandler, \
           3manager.org.apache.juli.FileHandler, \
           java.util.logging.ConsoleHandler

.handlers = 1catalina.org.apache.juli.FileHandler,
java.util.logging.ConsoleHandler

############################################################
# Handler specific properties.
# Describes specific configuration info for Handlers.
############################################################

1catalina.org.apache.juli.FileHandler.level = FINE
1catalina.org.apache.juli.FileHandler.directory =
${catalina.base}/logs
1catalina.org.apache.juli.FileHandler.prefix = catalina.
1catalina.org.apache.juli.FileHandler.maxDays = 90
1catalina.org.apache.juli.FileHandler.encoding = UTF-8

2localhost.org.apache.juli.FileHandler.level = FINE
2localhost.org.apache.juli.FileHandler.directory =
${catalina.base}/logs
2localhost.org.apache.juli.FileHandler.prefix = localhost.
2localhost.org.apache.juli.FileHandler.maxDays = 90
2localhost.org.apache.juli.FileHandler.encoding = UTF-8

3manager.org.apache.juli.FileHandler.level = FINE
3manager.org.apache.juli.FileHandler.directory =
${catalina.base}/logs
3manager.org.apache.juli.FileHandler.prefix = manager.
3manager.org.apache.juli.FileHandler.bufferSize = 16384
3manager.org.apache.juli.FileHandler.maxDays = 90
3manager.org.apache.juli.FileHandler.encoding = UTF-8

java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter =
java.util.logging.OneLineFormatter
java.util.logging.ConsoleHandler.encoding = UTF-8

############################################################
# Facility specific properties.
# Provides extra control for each logger.
############################################################

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].level =
INFO
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].handler
s = \
   2localhost.org.apache.juli.FileHandler

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manag
er].level = INFO
```

```
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manag
er].handlers = \
   3manager.org.apache.juli.FileHandler

# For example, set the org.apache.catalina.util.LifecycleBase logger
to log
# each component that extends LifecycleBase changing state:
# org.apache.catalina.util.LifecycleBase.level = FINE
```

Пример logging.properties для сервлет-примеров веб-приложения, помещаемых в WEB-INF/classes внутри веб-приложения:

```
handlers = org.apache.juli.FileHandler,
java.util.logging.ConsoleHandler

############################################################
# Handler specific properties.
# Describes specific configuration info for Handlers.
############################################################

org.apache.juli.FileHandler.level = FINE
org.apache.juli.FileHandler.directory = ${catalina.base}/logs
org.apache.juli.FileHandler.prefix = ${classloader.webappName}.

java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter =
java.util.logging.OneLineFormatter
```

6.2.1. Справочные документы

Для получения дополнительной информации см. ссылки на документацию в Приложении 1.

6.2.2. Вопросы промышленной эксплуатации

Обратите внимание на следующие советы и приемы:

- Подумайте над удалением ConsoleHandler из конфигурации. По умолчанию (благодаря настройке .handlers) регистрация происходит как в FileHandler, так и в ConsoleHandler. Результат последней обычно фиксируется в файле, таком как catalina.out. Таким образом получается две копии одних и тех же сообщений.
- Рассмотрите удаление FileHandler для приложений, которые вы не используете. Например, того, который предназначен для host-manager.
- По умолчанию обработчики используют для записи файлов журнала системное кодирование по умолчанию. Оно может быть сконфигурировано со свойством encoding. Подробности см. в Javadoc.
- Рассмотреть конфигурирование Access log.

7. Расширенный ввод/вывод и Libercat Certified

ПРИМЕЧАНИЕ: Для использования этих функций необходимо использовать HTTP коннекторы. AJP коннекторы эту функцию не поддерживают.

7.1. Асинхронная запись

При использовании HTTP-коннекторов (основанных на APR или NIO/NIO2), Libercat Certified поддерживает использование sendfile для отправки больших статических файлов. Этот процесс записи будет выполняться асинхронно оптимальным способом при увеличении нагрузки на систему. Вместо отправки обширного ответа с использованием блокировки записи становится возможным записывать содержимое в статический файл, используя код sendfile. Кэширующий вентиль в данном случае имеет преимущество, так как данные ответа сохраняются в файл кэша, а не хранятся в памяти. Поддержка sendfile доступна, если запрашиваемый атрибут `org.apache.tomcat.sendfile.support` установлен в `Boolean.TRUE`.

Любой сервлет может дать указание Libercat Certified на вызов sendfile с помощью настройки подходящих атрибутов запроса. Также необходимо установить корректное значение длины содержимого для ответа. При использовании sendfile предпочтительнее убедиться, что ни запрос, ни ответ не были обернуты, так как тело ответа будет отправлено позднее самим коннектором, оно не может быть отфильтровано. Сервлет может установить только 3 требуемых атрибута запроса и не должен отправлять никаких данных в ответ, но может использовать любой метод, который в результате модифицирует заголовок ответа (например, выполнить настройку cookies).

- `org.apache.tomcat.sendfile.filename`: Каноническое имя файла, которое будет отправлено как Строка
- `org.apache.tomcat.sendfile.start`: Начальное смещение как Long
- `org.apache.tomcat.sendfile.end`: Конечное смещение как Long

В дополнение к настройке данных параметров необходимо установить заголовок длины содержимого. Libercat Certified не сделает это за вас, поскольку вы могли уже записать данные в поток вывода.

Стоит отметить, что использование sendfile отменит любое сжатие, которое могло быть применено Libercat Certified к ответу.

8. Руководство по WebSocket

Libercat Certified поддерживает WebSocket, как описано [RFC 6455](https://tools.ietf.org/html/rfc6455).

8.1. Разработка приложения

Libercat Certified реализует Java WebSocket 1.1 API, как описано [JSR-356](https://www.jcp.org/en/jsr/detail?id=356).

Есть несколько примеров приложений, демонстрирующих использование WebSocket API. Вам нужно будет рассмотреть как сторону клиента [HTML](https://github.com/apache/tomcat/tree/9.0.x/webapps/examples/websocket), так и сервера [code](https://github.com/apache/tomcat/tree/9.0.x/webapps/examples/WEB-INF/classes/websocket).

8.2. Специальная конфигурация Libercat Certified WebSocket

Libercat Certified предоставляет несколько опций настройки, специфичных для WebSocket. Считается, что они со временем будут воплощены в спецификации WebSocket.

Задержка записи используется при передаче сообщений WebSocket в режиме блокировки, по умолчанию 20 000 миллисекунд (20 секунд). Это можно изменить установкой параметра `org.apache.tomcat.websocket.BLOCKING_SEND_TIMEOUT` в пользовательских настройках, прикрепленных к сессии WebSocket. Значение данного свойства должно быть `Long` и отражать задержку в миллисекундах. Для бесконечной задержки используйте `-1`.

В дополнение к методу `Session.setMaxIdleTimeout (long)`, который является частью Java WebSocket API, Libercat Certified обеспечивает дополнительный контроль тайм-аута сеанса из-за отсутствия активности. Установка свойства `org.apache.tomcat.websocket.READ_IDLE_TIMEOUT_MS` в коллекции свойств пользователя, прикрепленной к сеансу WebSocket, вызовет тайм-аут сеанса, если сообщение WebSocket не получено в течение указанного количества миллисекунд. Установка свойства `org.apache.tomcat.websocket.WRITE_IDLE_TIMEOUT_MS` вызовет тайм-аут сеанса, если в течение указанного количества миллисекунд сообщение не будет отправлено. Их можно использовать отдельно или вместе с `Session.setMaxIdleTimeout (long)`. Если свойство не указано, будет применяться тайм-аут простоя чтения и / или записи.

Если приложение не определяет `MessageHandler.Partial` для входящих двоичных сообщений, любые входящие двоичные сообщения должны помещаться в буфер так, чтобы сообщение целиком могло быть доставлено как один вызов и зарегистрировано `MessageHandler.Whole` для двоичных сообщений. Размер буфера по умолчанию для двоичных сообщений составляет 8192 байта. Это можно изменить

для веб-приложения с помощью настройки параметра инициализации контекста сервлета `org.apache.tomcat.websocket.binaryBufferSize` к желаемому значению в байтах.

Если приложение не определяет `MessageHandler.Partial` для входящих текстовых сообщений, любые входящие текстовые сообщения должны помещаться в буфер, чтобы все сообщение целиком могло быть доставлено как один вызов и зарегистрировано `MessageHandler.Whole` для текстовых сообщений. Размер буфера по умолчанию для текстовых сообщений составляет 8192 байта. Это можно изменить для веб-приложения с помощью настройки параметра инициализации контекста сервлета `org.apache.tomcat.websocket.textBufferSize` к желаемому значению в байтах.

Спецификация Java WebSocket 1.0 не предусматривает программное развертывание после старта подтверждения WebSocket первой конечной точкой. По умолчанию Libercat Certified продолжает поддерживать дополнительное программное развертывание. Это поведение контролируется параметром инициализации контекста сервлета `org.apache.tomcat.websocket.noAddAfterHandshake`. Это можно изменить установкой параметра `org.apache.tomcat.websocket.STRICT_SPEC_COMPLIANCE` системных свойств `true`, но любая явная настройка контекста сервлета всегда будет иметь приоритет.

При использовании клиента WebSocket для соединения с конечными точками сервера задержка операций ввода/вывода при установлении соединения контролируется `userProperties`, предоставленного `javax.websocket.ClientEndpointConfig`. Свойство `org.apache.tomcat.websocket.IO_TIMEOUT_MS` и задержка как `String` в миллисекундах. По умолчанию 5000 (5 секунд).

При использовании клиента WebSocket для соединения с безопасными конечными точками сервера конфигурация клиента SSL контролируется `userProperties`, предоставленного `javax.websocket.ClientEndpointConfig`. Поддерживаются следующие пользовательские настройки:

- `org.apache.tomcat.websocket.SSL_CONTEXT`
- `org.apache.tomcat.websocket.SSL_PROTOCOLS`
- `org.apache.tomcat.websocket.SSL_TRUSTSTORE`
- `org.apache.tomcat.websocket.SSL_TRUSTSTORE_PWD`

Пароль хранилища доверенных сертификатов по умолчанию – `changeit`.

Если свойство `org.apache.tomcat.websocket.SSL_CONTEXT` установлено, то свойства `org.apache.tomcat.websocket.SSL_TRUSTSTORE` и `org.apache.tomcat.websocket.SSL_TRUSTSTORE_PWD` будут игнорироваться.

Для безопасности конечных точек сервера верификация имени хоста включена по умолчанию. Для обхода верификации (не рекомендуется) необходимо обеспечить пользовательскую настройку `SSLContext` через свойство пользователя `org.apache.tomcat.websocket.SSL_CONTEXT`. Настраиваемый пользователем `SSLContext`

должен быть сконфигурирован при помощи настраиваемого пользователем `TrustManager`, расширяющего `javax.net.ssl.X509ExtendedTrustManager`. Желаемая верификация (или ее отсутствие) может затем контролироваться подходящими реализациями индивидуальных абстрактных методов.

При использовании клиента WebSocket для соединения с конечными точками сервера, количество перенаправленных запросов HTTP, которым следует клиент, контролируется `userProperties`, предоставленного `javax.websocket.ClientEndpointConfig`. Свойство `<ocde>org.apache.tomcat.websocket.MAX_REDIRECTIONS</ocde>`. Значение по умолчанию 20. Поддержку перенаправления можно отключить установкой значения в 0.

9. Вентиль перезаписи - Rewrite Valve

Вентиль перезаписи реализует функцию перезаписи URL-адреса аналогично функции mod_rewrite HTTP-сервера Apache.

9.1. Конфигурация

Вентиль перезаписи настраивается как вентиль с использованием имени класса `org.apache.catalina.valves.rewrite.RewriteValve`.

Вентиль перезаписи можно настроить как вентиль, добавленный в Host. См. ссылку на дополнительную документацию по virtual-server для получения информации по настройке. Будет использоваться файл `rewrite.config`, содержащий директивы перезаписи; он должен находиться в папке конфигурации Host.

Он также может быть в context.xml веб-приложения. Затем вентиль использует файл `rewrite.config`, содержащий директивы перезаписи; он должен находиться в папке WEB-INF веб-приложения.

9.2. Директивы

Файл rewrite.config содержит список директив, которые очень похожи на директивы, используемые mod_rewrite, в частности, центральную директиву RewriteRule и директиву RewriteCond. Строки, начинающиеся с символа `#`, считаются комментариями и игнорируются.

Примечание: Настоящий раздел является измененной версией документации mod_rewrite с авторским правом The Apache Software Foundation 1995–2006, лицензированной по лицензии Apache, Версия 2.0.

9.2.1. RewriteCond

Синтаксис: `RewriteCond TestString CondPattern`.

Директива RewriteCond определяет условия правил. Одна или несколько директив RewriteCond могут предшествовать директиве RewriteRule. Следующее правило затем используется, только если и текущее состояние универсального кода ресурса соответствует его шаблону, и соблюдаются эти условия.

TestString – это строка, которая помимо обычного текста может содержать следующие расширенные конструкции:

- **RewriteRule backreferences**: Это обратные ссылки вида **$N**(0 <= N <= 9), предоставляющие доступ к сгруппированным частям (в круглых скобках) шаблона, от RewriteRule, подчиняющегося текущему набору условий RewriteCond .
- **RewriteCond backreferences**: Это обратные ссылки вида **%N**(1 <= N <= 9), предоставляющие доступ к сгруппированным частям (опять же, в круглых скобках) шаблона, от последнего соответствующего RewriteCond в текущем наборе условий.
- **RewriteMap expansions**: Это расширения вида ${mapname:key|default}.
- **Server-Variables**: Это переменные вида %{ NAME_OF_VARIABLE }, где *NAME_OF_VARIABLE* может быть строкой, взятой из следующего списка:
    - **HTTP headers:**

      HTTP_USER_AGENT HTTP_REFERER HTTP_COOKIE HTTP_FORWARDED HTTP_HOST HTTP_PROXY_CONNECTION HTTP_ACCEPT
    - **connection & request:**

      REMOTE_ADDR REMOTE_HOST REMOTE_PORT REMOTE_USER REMOTE_IDENT REQUEST_METHOD SCRIPT_FILENAME REQUEST_PATH CONTEXT_PATH SERVLET_PATH PATH_INFO QUERY_STRING AUTH_TYPE
    - **server internals:**

      DOCUMENT_ROOT SERVER_NAME SERVER_ADDR SERVER_PORT SERVER_PROTOCOL SERVER_SOFTWARE
    - **date and time:**

      TIME_YEAR TIME_MON TIME_DAY TIME_HOUR TIME_MIN TIME_SEC TIME_WDAY TIME
    - **specials:**

      THE_REQUEST REQUEST_URI REQUEST_FILENAME HTTPS

Все эти переменные соответствуют аналогично названным заголовкам HTTP MIME и методам Servlet API. Большинство из них описаны в других местах *Руководства администратора* или в спецификации CGI. Ниже перечислены те, которые относятся только к вентилю перезаписи.

Переменная	Описание
REQUEST_PATH	Соответствует полному пути, используемому для сопоставления.
CONTEXT_PATH	Соответствуют пути сопоставленного контекста.
SERVLET_PATH	Соответствует пути сервлета.
THE_REQUEST	Полная строка HTTP-запроса, отправляемая от браузера к серверу (например, «`GET /index.html HTTP/1.1`»). Не включает в себя дополнительные заголовки, отправляемые браузером.
REQUEST_URI	Ресурс, запрашиваемый в строке HTTP-запроса. (в примере выше это «/index.html»).
REQUEST_FILENAME	Полный путь локальной файловой системы до файла или скрипта, соответствующего запросу.
HTTPS	Содержит текст «on» (вкл.), если в соединении используется SSL/TLS, или «off» (выкл.), если нет.

Необходимо учитывать следующее:

1. Переменные SCRIPT_FILENAME и REQUEST_FILENAME содержат одинаковое значение – значение поля `filename` внутренней структуры `request_rec` сервера Apache. Первое имя – это общеизвестное имя переменной CGI, а второе – соответствующий эквивалент REQUEST_URI (содержащий значение поля `uri` для `request_rec`).
2. `%{ENV:variable}`, где в качестве *variable* может быть любое свойство системы Java, также доступно.
3. `%{SSL:variable}`, где *variable* это имя переменной окружения SSL; реализовано, кроме `SSL_SESSION_RESUMED`, `SSL_SECURE_RENEG`, `SSL_COMPRESS_METHOD`, `SSL_TLS_SNI`, `SSL_SRP_USER`, `SSL_SRP_USERINFO`, `SSL_CLIENT_VERIFY`, `SSL_CLIENT_SAN_OTHER_msUPN_n`, `SSL_CLIENT_CERT_RFC4523_CEA`, `SSL_SERVER_SAN_OTHER_dnsSRV_n`. Когда используется OpenSSL, переменные, связанные с сертификатом сервера, с префиксом `SSL_SERVER_`, недоступны. Пример: `%{SSL:SSL_CIPHER_USEKEYSIZE}` может быть расширено до `128`.
4. `%{HTTP:header}`, где в качестве *header* может быть любое имя MIME-заголовка HTTP; может использоваться в любой момент для получения значения заголовка, отправленного в HTTP-запросе. Пример: `%{HTTP:Proxy-Connection}` – это значение HTTP-заголовка «`Proxy-Connection:`».

CondPattern – это шаблон условия, регулярное выражение, которое применяется для текущего экземпляра TestString. Сначала оценивается TestString, после чего производится сопоставление с CondPattern.

CondPattern – это _регулярное выражение perl с некоторым добавлением:

1. Вы можете добавлять перед строкой шаблона символ «!» (восклицательный знак) для указания **non**несоответствующего шаблона.

2. Имеются некоторые особые варианты *CondPatterns*. Вместо реальных строк регулярного выражения вы также можете использовать одну из следующих строк:

    - «**<CondPattern**» (лексикографически предшествует) Воспринимает *CondPattern* как простую строку и лексикографически сравнивает с *TestString*. Истинное значение, если *TestString* лексикографически находится до *CondPattern*.
    - «**>CondPattern**» (лексикографически следует) Воспринимает *CondPattern* как простую строку и лексикографически сравнивает с *TestString*. Истинное значение, если *TestString* лексикографически находится после *CondPattern*.
    - «**=CondPattern**» (лексикографически равно) Воспринимает *CondPattern* как простую строку и лексикографически сравнивает с *TestString*. Истинное значение, если *TestString* лексикографически равно *CondPattern* (две строки абсолютно идентичны, все символы совпадают). Если *CondPattern* имеет значение**""** (две кавычки), происходит сравнение *TestString* с пустой строкой.
    - «**-d**» (папка) Воспринимает *TestString* как имя пути и проверяет, существует ли он; является папкой.
    - «**-f**» (это обычный файл) Воспринимает *TestString* как имя пути и проверяет, существует ли он; является обычным файлом.
    - «**-s**» (это обычный файл определенного размера) Воспринимает *TestString* как имя пути и проверяет, существует ли он; является обычным файлом с размером больше нуля.

      ПРИМЕЧАНИЕ:До всех этих проверок может также стоять восклицательный знак («!») для обнуления значения.

3. Вы также можете задать специальные флаги для *CondPattern* путем добавления **[*flags*]** как третьего аргумента для директивы `RewriteCond`, где *flags* – это список следующих флагов, разделяемый запятыми:

    - «**nocase|NC**» (без учета регистра) Это делает проверку без учета регистра – разница между «A–Z» и «a–z» будет игнорироваться и в расширенном *TestString*, и в *CondPattern*. Этот флаг действует только для сравнений *TestString* и *CondPattern*. Он не применяется для проверок вспомогательных запросов и файловой системы.
    - «**ornext|OR**» (**or** или следующее условие) Используйте это для объединения условий правила с локальным ИЛИ вместо неявного И. Типичный пример:

      ```
      RewriteCond %{REMOTE_HOST}  ^host1.*  [OR]
      RewriteCond %{REMOTE_HOST}  ^host2.*  [OR]
      RewriteCond %{REMOTE_HOST}  ^host3.*
      RewriteRule ...some special stuff for any of these hosts...
      ```

      Без этого флага вам придется написать пару условие/правило три раза.

Пример:

Для перезаписи домашней страницы сайта в соответствии с заголовком запроса «`User-Agent:`» вы можете использовать следующее:

```
    RewriteCond  %{HTTP_USER_AGENT}  ^Mozilla.*
    RewriteRule  ^/$                 /homepage.max.html  [L]

    RewriteCond  %{HTTP_USER_AGENT}  ^Lynx.*
    RewriteRule  ^/$                 /homepage.min.html  [L]

    RewriteRule  ^/$                 /homepage.std.html  [L]
```

Пояснение: Если вы используете браузер, определяющийся как «Mozilla» (включая Navigator, Mozilla и т. п.), вы получаете максимальный вариант домашней страницы (может включать фреймы и другие специальные параметры). Если вы используете браузер Lynx (на основе терминала), вы получаете минимальный вариант домашней страницы (может быть версией, предназначенной для облегченного просмотра текста). Если не применяется ни одно из этих условий (вы используете любой другой браузер или ваш браузер определяется как нестандартный), вы получаете стандартный вариант домашней страницы.

9.2.2. RewriteMap

Синтаксис: `RewriteMap name rewriteMapClassName optionalParameters`
`rewriteMapClassName` позволяет использовать специальные значения:

- int:toupper: преобразует передаваемые значения к прописным символам
- int:tolower: преобразует передаваемые значения к строчным символам
- int:escape: URL кодирует передаваемые значения
- int:unescape: URL декодирует передаваемые значения

Используются карты с применением интерфейса, который пользователи должны реализовать. Имя его класса – `org.apache.catalina.valves.rewrite.RewriteMap`, а его код:

```
package org.apache.catalina.valves.rewrite;

public interface RewriteMap {
    default String setParameters(String params...); // calls
setParameters(String) with the first parameter if there is only one
    public String setParameters(String params);
    public String lookup(String key);
}
```

Реализация указанного класса - в нашем примере выше `rewriteMapClassName` - будет создана и инициализирована с необязательным параметром - `optionalParameters` (будьте осторожны с пробелами) - путем вызова `setParameters`

`(String)`. Затем этот экземпляр будет зарегистрирован под именем, указанным в качестве первого параметра правила `RewriteMap`.

Примечание. Вы можете использовать более одного параметра. Они должны быть разделены пробелами. Параметры могут заключаться в кавычки `"`. Это позволяет использовать пробелы внутри параметров.

Этому экземпляру Map будет присвоен ключ, который настроен в соответствующем `RewriteRule` путем вызова `lookup (String)`. Ваша реализация может возвращать `null`, чтобы указать, что следует использовать значение по умолчанию, или для возврата значения замены.

Скажем, вы хотите реализовать функцию, которая преобразует все ключи поиска в верхний регистр. Начните с реализации класса, реализующего интерфейс `RewriteMap`.

```
package example.maps;

 import org.apache.catalina.valves.rewrite.RewriteMap;

 public class UpperCaseMap implements RewriteMap {

   @Override
   public String setParameters(String params) {
     // nothing to be done here
     return null;
   }

   @Override
   public String lookup(String key) {
     if (key == null) {
       return null;
     }
     return key.toUpperCase();
   }
 }
```

Скомпилируйте этот класс, запакуйте его в jar, и поместите этот jar в `${CATALINA_BASE}/lib`.

Сделав это, теперь вы можете определить Map с помощью директивы RewriteMap и в дальнейшем использовать ее в RewriteRule.

```
RewriteMap uc example.maps.UpperCaseMap
RewriteRule ^/(.*)$ ${uc:$1}
```

Теперь запрос к URL-адресу /index.html будет перенаправлен на /INDEX.HTML.

9.2.3. RewriteRule

Синтаксис: `RewriteRule Pattern Substitution`.

Директива RewriteRule является основным рабочим инструментом перезаписи. Директива может использоваться неоднократно, при этом каждый экземпляр определяет одно правило перезаписи. Порядок, в котором эти правила

определяются, важен – это порядок, в котором они будут применяться во время работы.

Шаблоном является регулярное выражение perl, применяющееся для действующего URL-адреса. «Current» – это значение URL-адреса, когда применяется это правило. Это может быть не изначально запрошенный URL-адрес, который мог уже соответствовать предыдущему правилу и был изменен.

Предупреждение: Из-за того, как осуществляется сопоставление регулярных выражений на Java, недостаточно хорошо сформулированные шаблоны регулярных выражений подвержены «катастрофическому возврату», который также известен как «отказ в обслуживании с использованием регулярных выражений» (или ReDoS). Поэтому при использовании шаблонов RewriteRule следует быть особо внимательным. В целом, сложно автоматически определить такие уязвимые регулярные выражения, поэтому рекомендуется немного почитать информацию на тему катастрофического возврата. Рекомендуется ознакомиться с [OWASP ReDoS guide](https://www.owasp.org/index.php/Regular_expression_Denial_of_Service_-_ReDoS).

Некоторые указания по синтаксису регулярных выражений:

Text:

- `.` Один любой символ
- [chars] Класс символа: Любой символ класса «chars»
- `[^`chars] Класс символа: Символ, не относящийся к классу «chars»
- text1`|`text2 Альтернатива: text1 или text2

Quantifiers:

- `?` 0 или 1 случай предшествующего текста
- `*` 0 или N случаев предшествующего текста (N > 0)
- `+` 1 или N случаев предшествующего текста (N > 1)

Grouping:

- `(`text`)` Группирование текста (используется либо для установки границ альтернативы, как указано выше, либо для обратных ссылок, где на N**-ную группу может стоять ссылка в действии RewriteRule как $N)

Anchors:

- `^` Якорь начала строки
- `$` Якорь конца строки

Escaping:

- `\` char выход из представленного char (например, для указания char «.[]()» *и т. п.*)

Для получения подробной информации по регулярным выражениям зайдите на справочную страницу по регулярным выражениям («[perldoc perlre](https://perldoc.perl.org/perlre.html)»).

В правилах символ отрицания («`!`») также доступен как возможный префикс шаблона. Это позволяет отвергнуть шаблон; например: «если текущий URL-адрес НЕ соответствует этому шаблону». Этим можно пользоваться в исключительных случаях, когда проще сделать соответствие отрицательному шаблону, или в качестве последнего правила по умолчанию.

Примечание: При использовании символа отрицания, чтобы отвергнуть шаблон, нельзя включать в этот шаблон сгруппированные части с символом подстановки. Это связано с тем, что, когда наблюдается несоответствие шаблона (т. е. срабатывает отрицание), содержимое для групп отсутствует. Таким образом, при использовании шаблонов с отрицанием нельзя применять `$N` в строке подстановки!

Подстановка правила перезаписи представляет собой строку, которая подставляется для изначального URL-адреса (или заменяет его), соответствующего шаблону. Помимо обычного текста, сюда могут входить

1. обратные ссылки (`$N`) на шаблон RewriteRule;
2. обратные ссылки (`%N`) на последний соответствующий шаблон RewriteCond;
3. серверные переменные в качестве тестовых строк условия правила (`%{VARNAME}`);
4. вызовы mapping-function `${mapname:key|default}`).

Обратные ссылки являются идентификаторами вида $`N (N=0..9)`, которые заменяются содержимым `N`-ной группы соответствующего шаблона. Серверные переменные те же, что и для TestString директивы `RewriteCond`. Функции сопоставления исходят от директивы `RewriteMap` и поясняются там же. Эти три типа переменных развернуты в вышеуказанном порядке.

Как уже было указано, все правила перезаписи применяются для подстановки (в порядке, в котором они определены в файле конфигурации). URL-адрес полностью заменяется completely replaced на подстановку, и процесс перезаписи продолжается, пока все правила не будут применены или не произойдет явная остановка флагом `L`.

Специальные символы `$` и `%` можно указывать, добавляя перед ними обратный слеш `\`.

Существует специальная строка подстановки «-», означающая: БЕЗ подстановки! Она полезна при установке правил перезаписи, которые исключительно соответствуют URL-адресам, но не заменяют их ни на что. Она широко используется вместе с флагом C (цепочка), чтобы применить несколько шаблонов до подстановки.

В отличие от более свежих версий mod_rewrite, вентиль перезаписи Libercat Certified не имеет автоматической поддержки абсолютных URL-адресов (чтобы указать абсолютные URL-адреса, необходимо использовать специальный флаг редиректа, см. ниже) или прямого обслуживания файлов.

В качестве дополнительной меры вы можете задать специальные флаги для подстановки, путем добавления `[ flags ]` как третьего аргумента для директивы `RewriteRule`. Flags – это список следующих флагов, разделяемый запятыми:

- «`chain|C`» (соединение со следующим правилом) Этот флаг соединяет текущее правило со следующим правилом (которое в свою очередь может быть соединено со следующим правилом, и так далее). Это имеет следующий эффект: если правило соответствует, обработка продолжается как обычно – флаг не оказывает воздействия. Если правило не соответствует, следующие связанные правила пропускаются. Например, это можно использовать для удаления части «.www» внутри набора правил для одной папки, когда нужно сделать внешний редирект (где части «.www» быть не должно!).

- «`cookie|CO=`*NAME*:*VAL*:*domain*[:*lifetime*[:*path*]]» (задать **co**okie) Это задает cookie-файл в браузере клиента. Имя cookie-файла задается через *NAME*, а значение – через *VAL*. Поле *domain* – это домен cookie-файла, например, «.apache.org», необязательный параметр *lifetime* – это срок действия cookie-файла в минутах, а необязательный параметр *path* – это путь к cookie-файлу

- «`env|E=`*VAR*:*VAL*» (установить переменную среды) Этот флаг принудительно задает атрибуту запроса с именем *VAR* значение *VAL*, где *VAL* может содержать обратные ссылки регулярных выражений (`$N` и `%N`), которые будут расширены. Вы можете использовать этот флаг несколько раз, чтобы задать несколько переменных.

- «**forbidden|F**» (принудительно запрещает URL-адрес) Это принудительно запрещает текущий URL-адрес – он сразу отправляет обратно HTTP-ответ 403 (ЗАПРЕЩЕНО). Используйте этот флаг вместе с RewriteConds для условной блокировки некоторых URL-адресов.

- «**gone|G**» (принудительно удаляет URL-адрес) Это принудительно удаляет текущий URL-адрес – он сразу отправляет обратно HTTP-ответ 410 (УДАЛЕН). Используйте этот флаг для отметки страниц, которые больше не существуют.

- «**host|H**=*Host*» (применение перезаписи для хоста) Вместо перезаписи URL-адреса переписывается виртуальный хост.

- «**last|L**» (последнее правило) Останавливает процесс перезаписи на этом моменте и не применяет больше никаких правил перезаписи. Это соответствует команде Perllast или команде break в C. Используйте этот флаг, чтобы перезаписываемый на данный момент URL-адрес больше не перезаписывался другими правилами. Например, используйте его для перезаписи URL-адреса корневого пути («/») на настоящий путь, *например*, «/e/www/».

- «**next|N**» (следующий этап) Перезапуск процесса перезаписи (начиная снова с первого правила перезаписи). В этот раз соответствующий URL-адрес больше не является изначальным, а является URL-адресом, выдаваемым по последнему правилу перезаписи. Это соответствует команде Perl next или команде continue в C. Используйте этот флаг для перезапуска процесса

перезаписи, чтобы сразу перейти к верхней части цикла. **Be careful not to create an infinite loop!**

- «**nocase|NC**» (без учета регистра) Это делает *шаблон* нечувствительным к регистру, игнорируя разницу между «A-Z» и «a-z», когда *шаблон* соответствует текущему URL-адресу.

- «**noescape|NE**» (без преобразования выходных данных универсального кода ресурсов) Этот флаг не дает вентилю перезаписи применить обычные правила изолирования универсального кода ресурса на результат перезаписи. Обычно специальные символы (например, «%», «$», «;» и т. п.) преобразуются в свои шестнадцатеричные эквиваленты («%25», «%24» и «%3B» соответственно); этот флаг не дает этому произойти. Это позволяет символам процента появляться в выходных данных, как в следующем примере:

  ```
  RewriteRule /foo/(.*) /bar?arg=P1\%3d$1 [R,NE]
  ```

  что переводит «`/foo/zed`» в безопасный запрос для «`/bar?arg=P1=zed`».

- «**qsappend|QSA**» (добавление строки запроса) Этот флаг принуждает механизм перезаписи добавить часть строки запроса из строки подстановки в существующую строку, вместо ее замены. Используйте это, если хотите добавить дополнительные данные в строку запроса с помощью правила перезаписи.

- «**redirect|R [=*code*]**» (принудительный редирект) Префикс *Substitution* chttp://thishost[:thisport]/(делающий новый URL-адрес универсальным кодом ресурса) для принудительного внешнего редиректа. Если параметр *code* не указан, выдается HTTP-ответ 302 (НАЙДЕНО, ранее ВРЕМЕННО ПЕРЕМЕЩЕНО). Если вы хотите использовать другие коды ответов в диапазоне 300–399, просто укажите соответствующий номер или используйте одно из следующих имен символов:temp (по умолчанию), permanent,seeother. Используйте это для правил, если нужно канонизировать URL-адрес и выдать его клиенту – для перевода «/~» в «/u/» или чтобы всегда добавлять слеш к/u/*user* и т. д. **ПРИМЕЧАНИЕ:**Если вы используете этот флаг, убедитесь, что в поле подстановки указан действующий URL-адрес! В противном случае будет произведен редирект на несуществующий адрес. Помните, что сам по себе этот флаг лишь добавляетhttp://thishost[:thisport]/ к URL-адресу, и перезапись продолжается. Обычно требуется остановка перезаписи в этом моменте с мгновенным редиректом. Для остановки перезаписи необходимо добавить флаг «L».

- «**skip|S**=*num*» (пропуск одного или нескольких следующих правил) Этот флаг принуждает механизм перезаписи пропустить следующие правила *num* в последовательности, если текущее правило совпадает. Используйте это для создания псевдоконструкций «если-тогда-иначе»: Последнее правило условия «тогда» становитсяskip=N, где N – это количество правил в условии «иначе». (Это не то же самое, что флаг «chain|C»!)

- «**type|T**=*MIME-type*» (принудительный тип MIME) Принудительное использование *типа MIME* для целевого файла. Это может использоваться для установки типа содержимого в зависимости от определенных условий.

Например, следующий фрагмент позволяет.php *отображать* файлы с помощью mod_php , если они вызываются с расширением .phps :

```
RewriteRule ^(.+\.php)s$ $1 [T=application/x-httpd-php-source]
```

10. CDI 2, JAX-RS и зависимые библиотеки

CDI и JAX-RS — это зависимости для многих API и библиотек. В данном руководстве описано, как добавить поддержку этих зависимостей в Tomcat с помощью двух дополнительных модулей из источников Libercat Certified.

10.1. Поддержка CDI 2

Поддержка CDI 2 обеспечивается дополнительным модулем `modules/owb`. Он упаковывает проект Apache OpenWebBeans и позволяет добавить поддержку CDI 2 в контейнер Libercat Certified. Для сборки модуль использует Apache Maven, при этом сборка не доступна в качестве бинарного файла, так как она собирается с применением нескольких публичных JAR-файлов.

Процесс добавления поддержки CDI выглядит следующим образом:

```
cd $TOMCAT_SRC/modules/owb
mvn clean && mvn package
```

Полученный JAR-файл, расположенный по адресу target/tomcat-owb-x.y.z.jar (где x.y.z зависит от версии Apache OpenWebBeans, использованной при сборке), необходимо поместить в папку `lib` установки Tomcat.

После этого поддержку CDI можно разрешить для всех веб-приложений в контейнере, добавив следующий слушатель в элемент `Server` файла `server.xml`:

```
<Listener
   className="org.apache.webbeans.web.tomcat.OpenWebBeansListener"
   optional="true"
   startWithoutBeansXml="false"
/>
```

Если загрузка CDI-контейнера не удастся, слушатель вызовет некритическую ошибку.

Также можно разрешить поддержку CDI для отдельных веб-приложений, добавив следующий слушатель в элемент `Server` файла `context.xml`:

```
<Listener

className="org.apache.webbeans.web.tomcat.OpenWebBeansContextLifecycl
eListener"
/>
```

10.2. Поддержка JAX-RS

Поддержка JAX-RS обеспечивается дополнительным модулем `modules/cxf`. Он упаковывает проект Apache CXF и позволяет добавить поддержку JAX-RS в отдельные веб-приложения. Для сборки модуль использует Apache Maven, при этом сборка не доступна в качестве бинарного файла, так как она собирается с применением нескольких публичных JAR-файлов. Поддержка этой зависимости зависит от поддержки CDI 2, которую перед этим необходимо было добавить в контейнер или на уровень веб-приложений.

Процесс добавления поддержки JAX-RS выглядит следующим образом:

```
cd $TOMCAT_SRC/modules/cxf
mvn clean && mvn package
```

Полученный JAR-файл, расположенный по адресу `target/tomcat-cxf-x.y.z.jar` (где x.y.z зависит от версии Apache CXF, использованной при сборке) необходимо поместить в папку `/WEB-INF/lib` требуемого веб-приложения.

Если поддержка CDI 2 разрешена на уровне контейнера, JAR-файл также можно поместить в папку `lib` Libercat Certified, но в этом случае в каждое приложение по отдельности нужно будет добавить декларацию CXF Servlet (обычно она загружается веб-фрагментом в JAR-файле).

Необходимо использовать следующий класс CXF Servlet: `org.apache.cxf.cdi.CXFCdiServlet`. Его нужно добавить в соответствующий корневой путь, по которому будут доступны ресурсы JAX-RS.

10.3. Поддержка Eclipse Microprofile

Вы можете использовать артефакты ASF, которые реализуют спецификации Eclipse Microprofile посредством расширений CDI 2. После добавления поддержки CDI 2 и JAX-RS они могут использоваться отдельными веб-приложениями.

В качестве Maven-артефактов можно использовать следующие реализации (`org.apache.tomee.microprofile.TomEEMicroProfileListener`). Их нужно добавить в папку `/WEB-INF/lib` веб-приложений:

- **Конфигурация**:

  Maven-артефакт: `org.apache.geronimo.config:geronimo-config`

  Класс расширения CDI: `org.apache.geronimo.config.cdi.ConfigExtension`

- **Устойчивость к ошибкам**:

  Maven-артефакт: `org.apache.geronimo.safeguard:safeguard-parent`

  Класс расширения CDI: `org.apache.safeguard.impl.cdi.SafeguardExtension`

- **Состояние системы**:

Maven-артефакт: `org.apache.geronimo:geronimo-health`

Класс расширения CDI:
`org.apache.geronimo.microprofile.impl.health.cdi.GeronimoHealthExtension`

- **Метрики**:

  Maven-артефакт: `org.apache.geronimo:geronimo-metrics`

  Класс расширения CDI:
  `org.apache.geronimo.microprofile.metrics.cdi.MetricsExtension`

- **OpenTracing**:

  Maven-артефакт: `org.apache.geronimo:geronimo-opentracing`

  Класс расширения CDI:
  `org.apache.geronimo.microprofile.opentracing.microprofile.cdi.OpenTracing Extension`

- **OpenAPI**:

  Maven-артефакт: `org.apache.geronimo:geronimo-openapi`

  Класс расширения CDI:
  `org.apache.geronimo.microprofile.openapi.cdi.GeronimoOpenAPIExtension`

- **Rest-клиент**:

  Maven-артефакт: `org.apache.cxf:cxf-rt-rs-mp-client`

  Класс расширения CDI:
  `org.apache.cxf.microprofile.client.cdi.RestClientExtension`

- **Веб-токены JSON**: Примечание: в ознакомительных целях, не применимо вне Libercat Certified EE;

  Maven-артефакт: `org.apache.tomee:mp-jwt`

  Класс расширения CDI:
  `org.apache.tomee.microprofile.jwt.cdi.MPJWTCDIExtension`

11. AOT-компиляция

Libercat Certified поддерживает инструмент сборки нативного образа GraalVM для создания нативного бинарного файла, включая контейнер. В данном разделе документации описывается процесс сборки такого образа.

11.1. Установка

Инструмент для создания нативного образа гораздо проще использовать с отдельными JAR-файлами. В результате будет использован процесс упаковки JAR-файла с применением плагина Maven shade plugin (толстый JAR). Цель — создать

один JAR-файл, который будет содержать все необходимые классы из Tomcat, веб-приложений, а также дополнительные зависимости. Хотя в Tomcat были внедрены патчи для поддержки нативных образов, любая библиотека может оказаться несовместимой, из-за чего потребуется переписать код приложения (в документации GraalVM это прописано более подробно).

Скачайте и установите GraalVM или Mandrel.

Если вы используете GraalVM, первым шагом будет добавление инструмента для создания нативного образа.

```
export JAVA_HOME=/absolute...path...to/graalvm-ce-javaX-x.y.z
cd $JAVA_HOME/bin
./gu install native-image
```

Mandrel уже включает готовый к использованию инструмент для создания нативного образа, поэтому этот шаг можно пропустить. Нужно только настроить JAVA_HOME к папке, содержащей папку bin с бинарными файлами JVM:

```
export
JAVA_HOME=/absolute...path...to/mandrel-javaXX-platform-x.x.x.x/mandr
elJDK
```

Скачайте модуль Tomcat Stuffed с сайта разработчика https://github.com/apache/tomcat/tree/main/modules/stuffed и поместите все файлы в папку `stuffed`.

```
export TOMCAT_STUFFED=/absolute...path...to/stuffed
```

Для сборки понадобится как Ant, так и Maven.

11.2. Упаковка и сборка

Структура каталога в папке tomcat-stuffed такая же, как в обычном проекте Tomcat. Основные конфигурационные файлы находятся в папке conf, если вы используете файл server.xml по умолчанию, веб-приложения расположены в папке webapps.

Первый шаг — сборка толстого JAR-файла Tomcat со всеми зависимостями. Любой JSP в веб-приложении нужно предварительно скомпилировать и упаковать.

```
cd $TOMCAT_STUFFED
mvn package
ant -Dwebapp.name=somewebapp -f webapp-jspc.ant.xml
```

Теперь необходимо добавить зависимости для веб-приложения в основной pom.xml, а после этого собрать толстый JAR-файл.

```
mvn package
```

Так как в случае AOT-компиляции необходимо по возможности избегать использования рефлексии, можно сгенерировать и компилировать код Tomcat Embedded из основного конфигурационного файла server.xml и файлов context.xml, используемых для конфигурации контекстов.

```
$JAVA_HOME/bin/java\
    -Dcatalina.base=.\
    -Djava.util.logging.config.file=conf/logging.properties\
    -jar target/tomcat-stuffed-1.0.jar --catalina -generateCode
src/main/java
mvn package
```

Подразумевается, что перед выполнением оставшегося описанного здесь процесса вы выполните этот этап, и аргументы `--catalina -useGeneratedCode` будут добавлены в командную строку. В противном случае их необходимо удалить.

11.3. Конфигурация нативного образа

Нативные образы не поддерживают динамическую загрузку классов или рефлекцию, если только это явно не указано в описателе. При их генерации используется трассирующий агент из GraalVM, поэтому в некоторых случаях требуется дополнительная конфигурация вручную.

Запустите Substrate VM GraalVM с помощью трассирующего агента:

```
$JAVA_HOME/bin/java\

-agentlib:native-image-agent=config-output-dir=$TOMCAT_STUFFED/target
/\
  -Dorg.graalvm.nativeimage.imagecode=agent\
  -Dcatalina.base=.
-Djava.util.logging.config.file=conf/logging.properties\
  -jar target/tomcat-stuffed-1.0.jar --catalina -useGeneratedCode
```

Теперь доступ ко всем путям, которые ведут из веб-приложения к динамической загрузке классов (например, доступ к сервлету, websocket и т.д.), осуществляется через скрипт, выполняющий веб-приложение. Сервлеты можно загрузить при запуске без осуществления фактического доступа. Также для загрузки дополнительных классов при запуске можно использовать слушатели.

Агент сгенерировал описания. На данном этапе нужно добавить объекты, которые не отслеживаются, включая базовые интерфейсы, комплекты ресурсов, рефлексию на базе BeanInfo и т.д. Более подробную информацию об этом процессе вы можете найти в документации Graal.

11.4. Сборка нативного образа

Если все было сделано правильно, с помощью инструмента для создания нативного образа теперь можно собрать нативный образ.

```
$JAVA_HOME/bin/native-image --no-server\
  --allow-incomplete-classpath --enable-https\

--initialize-at-build-time=org.eclipse.jdt,org.apache.el.parser.Simpl
```

```
eNode,javax.servlet.jsp.JspFactory,org.apache.jasper.servlet.JasperIn
itializer,org.apache.jasper.runtime.JspFactoryImpl\
  -H:+JNI -H:+ReportUnsupportedElementsAtRuntime\
  -H:+ReportExceptionStackTraces
-H:EnableURLProtocols=http,https,jar,jrt\
  -H:ConfigurationFileDirectories=$TOMCAT_STUFFED/target/\

-H:ReflectionConfigurationFiles=$TOMCAT_STUFFED/tomcat-reflection.jso
n\
  -H:ResourceConfigurationFiles=$TOMCAT_STUFFED/tomcat-resource.json\
  -H:JNIConfigurationFiles=$TOMCAT_STUFFED/tomcat-jni.json\
  -jar $TOMCAT_STUFFED/target/tomcat-stuffed-1.0.jar
```

Дополнительный параметр `--static` обеспечивает статическое связывание glibc, zlib и libstd++ в созданном бинарном файле.

Запуск нативного образа:

```
./tomcat-stuffed-1.0 -Dcatalina.base=.
-Djava.util.logging.config.file=conf/logging.properties --catalina
-useGeneratedCode
```

11.5. Совместимость

Нативный образ изначально поддерживает сервлеты, JSP, EL, websocket, контейнер Tomcat, tomcat-native, HTTP/2.

На момент составления этой документации, библиотека JULI не поддерживалась Graal в качестве конфигурационного параметра log manager; также отмечались проблемы со статическим инициализатором, поэтому вместо этого нужно использовать стандартные логгеры и имплементацию java.util.logging.

Если вы используете файл server.xml по умолчанию, из конфигурации необходимо удалить некоторые серверные обработчики событий, так как они не совместимы с нативными образами, например, слушатель JMX (JMX не поддерживается) и слушатели для предотвращения утечки памяти (использование внутреннего кода, отсутствующего в Graal).

Отсутствующие объекты для лучшей работы Tomcat:

- JMX: обычный мониторинг и контроль не применим
- java.util.logging LogManager: конфигурация посредством системного свойства не реализуется, поэтому вместо JULI необходимо использовать стандартную java.util.logging
- Конфигурация статического связывания: tomcat-native нельзя статически связать

Приложение 1.

Перечень ссылок на дополнительную документацию

1. Libercat Certified JDBC pool (https://tomcat.apache.org/tomcat-9.0-doc/jdbc-pool.html)
2. Context (https://tomcat.apache.org/tomcat-9.0-doc/config/context.html)
3. Valves (https://tomcat.apache.org/tomcat-9.0-doc/config/valve.html#Access_Logging)
4. System properties (https://tomcat.apache.org/tomcat-9.0-doc/config/systemprops.html#Logging)
5. Javadoc для пакета org.apache.juli (https://tomcat.apache.org/tomcat-9.0-doc/api/org/apache/juli/package-summary.html).
6. Oracle Java 8 Javadoc для пакета java.util.logging (https://docs.oracle.com/javase/8/docs/api/java/util/logging/package-summary.html)
7. Access log (https://tomcat.apache.org/tomcat-9.0-doc/config/valve.html#Access_Logging)
8. virtual-server (https://tomcat.apache.org/tomcat-9.0-doc/config/host.html)
9. Документация по RewriteMap (https://tomcat.apache.org/tomcat-9.0-doc/rewrite.html#mapfunc)
10. mapping-function (https://tomcat.apache.org/tomcat-9.0-doc/rewrite.html#RewriteMap)

Лист регистрации изменений									
Номера листов (страниц)					*Всего листов (страниц) в докум.*	*№ документа*	*Входящий № сопроводительного докум. и дата*	*Подп.*	*Дата*
Изм	*измененных*	*замененных*	*новых*	*аннулированных*					
1	1	-	-	-	51	РОФ.06262534.00036-01 90-99 01			