Структура веб-приложения на платформе Java EE

Веб-приложение включает в себя множество ресурсов, в том числе сервлеты, JSP-страницы, вспомогательные классы, сторонние библиотеки в виде JAR-файлов, HTML-файлы и т.д. Данные ресурсы могут быть взаимосвязаны, например, сервлет может использовать классы из сторонней библиотеки или перенаправлять запрос на JSP-страницу. Следовательно, одним ресурсам необходимо знать о расположении других ресурсов. Кроме того, веб-приложение должно быть переносимым между различными веб-контейнерами. Для решения данных задач в спецификации сервлетов определен стандартный способ упаковки веб-приложений.

Ресурсы веб-приложения хранятся в иерархии каталогов, структура и назначение которых строго определены. Рассмотрим структуру каталогов на примере веб-приложения `helloapp`. Каталог `helloapp` представляет собой корень документов (document root) веб-приложения `helloapp`. Запрос ресурса `http://www.myserver.com/helloapp/index.html` относится к файлу `index.html` в каталоге `helloapp`. (при условии, что корень данного веб-приложения – `/helloapp`) Все находящиеся в открытом доступе файлы следует размещать в данном каталоге. Как правило, данные файлы хранятся в нескольких подкаталогах по типу или назначению.

```
|- helloapp
      |- html (содержит все HTML-файлы)
      |- jsp (содержит все JSP-файлы)
      |- images (содержит все GIF-, JPEG-, BMP-файлы)
      |- javascripts (содержит все *.js-файлы)
      |- index.html (HTML-файл, возвращаемый по умолчанию)
      |- WEB-INF
            |- classes
            |- lib
            |- web.xml
```

В каждом веб-приложении непосредственно в корневом каталоге должен размещаться каталог `WEB-INF`. Несмотря на то, что физически данный каталог размещается в корне документов, он не считается частью этого каталога, и располагающиеся в нем файлы недоступны по запросу. В каталоге `WEB-INF` содержатся:

- Каталог `classes`, в котором размещаются файлы классов сервлетов и вспомогательных классов, не включенные в какой-либо JAR-файл. Файлы классов должны размещаться в каталогах, соответствующих структуре пакетов. Во время выполнения веб-контейнер добавляет каталог `classes` в путь поиска классов (classpath) для данного веб-приложения.
- Каталог `lib`, в котором размещаются все используемые приложением JAR/ZIP-файлы. Например, в данный каталог следует поместить JAR-файл с JDBC-драйвером. Классы сервлетов можно упаковать в JAR-файл и также разместить в данном каталоге. Во время выполнения веб-контейнер добавляет все JAR/ZIP-файлы из каталога `lib` в путь поиска классов для данного веб-приложения.
- Файл `web.xml` – дескриптор установки, содержащий основные установки веб-приложения, такие как объявления сервлетов и путей к ним, параметры инициализации, ограничения безопасности и т.п.

Для упрощения установки веб-приложения и его перемещения из одной среды в другую оно может быть упаковано в WAR-файл. Данный вид архива отличается от JAR-файла только порядком его обработки веб-контейнером. Например, если разместить WAR-файл в каталог `webapps` веб-контейнера Apache Tomcat, то контейнер автоматически разархивирует его в подкаталог, название которого совпадает с названием WAR-файла, и

запустит соответствующее веб-приложение. Любой веб-контейнер может установить веб-приложение, упакованное в WAR-файл.

Сопоставление запросов веб-компонентам.

При сопоставлении запросов веб-компонентам используются имена статических файлов и псевдонимы компонентов, указываемые в установочном дескрипторе. Для одного компонента можно указать несколько псевдонимов. При формировании псевдонима можно использовать `*`.

При получении запроса от клиента веб-контейнер определяет веб-приложение, которое должно его обработать. Выбирается приложение, контекст которого совпадает с наиболее длинным участком от начала запрашиваемого URL. Например, если имеется два веб-приложения с контекстами `/admin` и `/admin/console`, то запрос по адресу `/admin/console/monitor/index.jsp` будет передан на обработку во второе приложение. Совпавший участок URL считается контекстом веб-приложения (context path) и не учитывается при определении компонента, обрабатывающего запрос.

Определение обрабатывающего запрос компонента выполняется по URL запроса за вычетом из него контекста веб-приложения и параметров по следующему алгоритму (используется первое удачное совпадение):

1. Поиск точного совпадения пути запроса с путем (псевдонимом) сервлета.
2. Рекурсивный поиск в пути запроса наиболее длинного префикса, соответствующего псевдониму сервлета. Веб-контейнер поочередно выделяет подкаталоги, используя символ `/` как разделитель, и сравнивает путь к подкаталогу к псевдонимами сервлетов.
3. Если последний сегмент в пути запроса содержит расширение (например, `.jsp`), то поиск сервлета, обрабатывающего запросы для данного расширения.
4. Если для приложения определен так называемый «сервлет по умолчанию» (default servlet), то он используется для обработки запроса.

Для определения псевдонимов для веб-компонентов в установочном дескрипторе используется следующий синтаксис:

- Строка, начинающаяся с символа `/` и заканчивающаяся суффиксом `/*`, используется для сопоставления по подстроке на шаге 2 алгоритма.
- Строка, начинающаяся с префикса `*.`, используется для сопоставления по расширению на шаге 3.
- Строка, состоящая из символа `/`, определяет для приложения сервлет по умолчанию.
- Любые другие строки используются только для поиска точного сопадения на шаге 1.

Пример 1. Если в установочном дескрипторе задано следующее соответствие:

Псевдоним	Компонент
`/foo/bar/*`	servlet1
`/baz/*`	servlet2
`/catalog`	servlet3
`*.bop`	servlet4

, то на этапе обработки запросов возможны такие ситуации:

URL запроса	Компонент, обрабатывающий запрос
`/foo/bar/index.html`	servlet1
`/foo/bar/index.bop`	servlet1
`/baz`	servlet2
`/baz/index.html`	servlet2
`/catalog`	servlet3
`/catalog/index.html`	Default servlet
`/catalog/rarecar.bop`	servlet4
`/index.bop`	servlet4

В установочном дескрипторе можно указать страницы или компоненты, которые будут обрабатывать запросы в случае, если эти запросы относятся не к конкретным страницам или компонентам, а к каталогам веб-модуля. Это так называемые страницы приветствия (welcome files). В сервере приложений Sun по умолчанию страницы приветствия – `index.html`, `index.jsp`. Если URL запроса относится к каталогу, то он дополняется именем страницы по умолчанию, и если она существует, то она и будет генерировать ответ, иначе для обработки запроса будет использован сервлет по умолчанию. Если сервлет по умолчанию не определен, то будет отображено содержимое каталога, к которому относятся запросы, или сгенерировано стандартное сообщение об ошибке 404 (ресурс не найден, Resource Not Found).

Пример 2. В веб-модуле существует такое содержимое:

```
/foo/index.html
/foo/default.jsp
/foo/orderform.html
/foo/home.gif
/catalog/default.jsp
/catalog/products/shop.jsp
/catalog/products/register.jsp
```

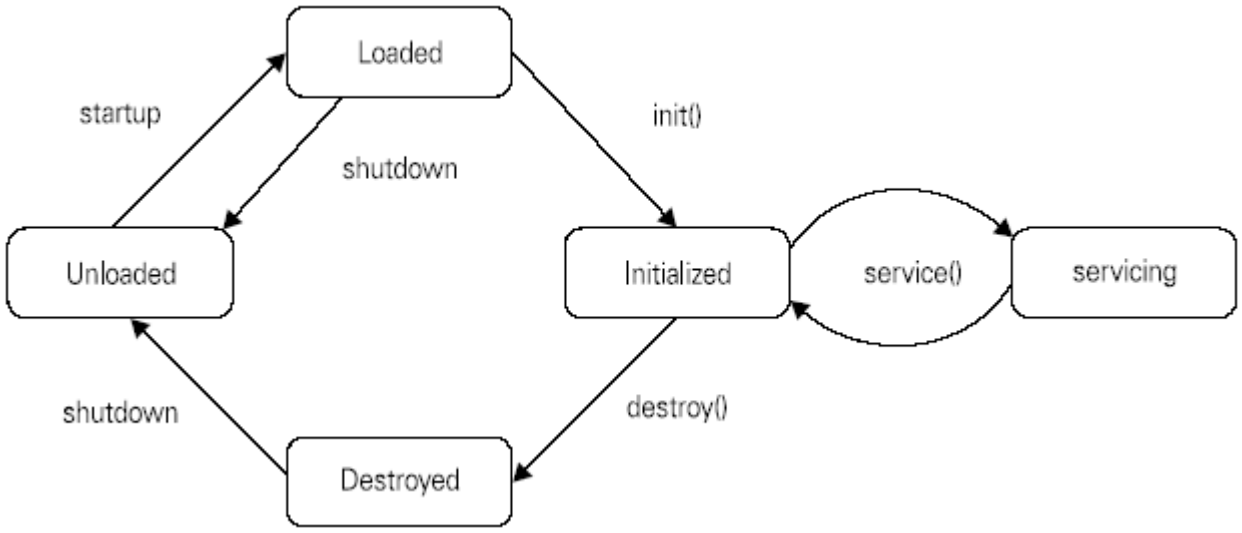
В установочном дескрипторе для него указаны следующие страницы приветствия: `index.html`, `default.jsp`. На этапе обработки запросов возможны такие ситуации:

URL запроса	Страница, которая будет использована
`/foo`	`/foo/index.html`
`/foo/`	`/foo/index.html`
`/catalog`	`/catalog/default.jsp`
`/catalog/`	`/catalog/default.jsp`
`/catalog/index.html`	Ошибка 404
`/catalog/products`	`/catalog/products` или Default servlet или Ошибка 404

Сервлеты

Сервлеты являются одним из видов веб-компонентов и предназначены для реализации функциональности, которая может быть активирована по веб-протоколам (HTTP, HTTP over SSL). Взаимодействие пользователя с сервлетом ведется через веб-браузер.

С точки зрения платформы Java EE сервлет – это класс, который реализует интерфейс `javax.servlet.Servlet`. Все классы, интерфейсы и исключения, относящиеся к технологии сервлетов, сгруппированы в пакете `javax.servlet`. Средства этого пакета не содержат никакой специфики, относящейся к протоколу взаимодействия клиента с сервером, и позволяют разрабатывать сервлеты, не зависящие от протокола. Средства, учитывающие специфику протокола HTTP, который является стандартом де факто для взаимодействия клиента с сервером, сгруппированы в пакете `javax.servlet.http`.



Жизненный цикл сервлета можно представить так. Состояние A – экземпляр сервлета не существует, состояние B – экземпляр существует и готов обработать запрос. Переход 1 – веб-контейнер создает объект класса сервлета и вызывает его метод `init()`. Переход 2 – сервлет обрабатывает поступивший к нему запрос, для чего веб-контейнер вызывает метод `service()` экземпляра сервлета. Переход 3 – веб-контейнер уничтожает экземпляр сервлета, вызывая перед этим метод `destroy()`. Создание и уничтожение экземпляра сервлета выполняется автоматически веб-контейнером. Зачастую это происходит при запуске (создании) и останове (уничтожении) веб-контейнера. В ходе работы экземпляр сервлета может быть на время уничтожен при сравнительно малой нагрузке на него и сравнительно большой нагрузке на другие сервлеты в условиях нехватки ресурсов.

Рассмотрим подробнее методы интерфейса `Servlet`.

1) `void init( ServletConfig config ) throws ServletException, IOException`

Разработчик сервлета реализует этот метод для инициализации создаваемого экземпляра сервлета. Контейнер в качестве параметра передает сервлету объект типа `ServletConfig`, который используется сервлетом для извлечения конфигурационных параметров. Эти параметры могут быть заданы в дескрипторе установки (deployment descriptor) и позволяют настраивать поведение сервлета без его перекомпиляции. Кроме того, этот объект используется для получения других объектов инфраструктуры, поддерживаемой веб-контейнером для сервлета.

При реализации метода `init()` важно сохранить полученный объект типа `ServletConfig` в поле класса сервлета, что позволит использовать его в методе `service()` при обработке запросов.

Также в методе `init()` реализуют поиск и сохранение в полях класса сервлета объектов из внутреннего пространства имен сервера приложений. Наиболее типичный случай – это поиск источников данных (или других фабрик соединений с менеджерами ресурсов) и EJB-компонентов.

2) `void service( ServletRequest request, ServletResponse response ) throws ServletException, IOException`

Веб-контейнер вызывает метод `service()` для обработки каждого запроса, поступившего к сервлету. Основная задача сервлета – сформировать ответ на запрос. При этом веб-контейнер автоматически создает два объекта: первый – типа `ServletRequest`, а второй – типа `ServletResponse`, и передает их как параметры методу `service()`. Первый объект позволяет извлечь всю информацию, пришедшую вместе с запросом, а второй – сформировать ответ. По окончании работы метода `service()` обработка запроса сервлетом считается законченной, после чего использовать переданные объекты `request` и `response` нельзя.

3) `void destroy()`

В данном методе разработчик должен освободить все ресурсы, которые были получены в методе `init()`. Зачастую реализация данного метода пустая.

4) `ServletConfig getServletConfig()`

Данный метод должен возвращать тот самый объект типа `ServletConfig`, который был передан методу `init()` в качестве параметра. Этот метод используется для удобства при расширении стандартных базовых классов сервлетов.

5) `String getServletInfo()`

Данный метод должен возвращать строку с кратким описанием сервлета.

При разработке собственных сервлетов обычно не реализуют непосредственно интерфейс `Servlet`, а наследуют его реализацию по умолчанию, расширяя один из двух стандартных базовых классов для сервлетов: `javax.servlet.GenericServlet` или `javax.servlet.http.HttpServlet`.

Первый из них предназначен для разработки сервлетов, независящих от протокола взаимодействия с клиентами. При расширении класса `GenericServlet` для обработки запросов переопределяют его метод `service()`.

Класс `HttpServlet` расширяет `GenericServlet`, дополняя его средствами, специфичными для протокола HTTP – именно он используется как базовый в большинстве случаев. Реализация метода `service()` в классе `HttpServlet` обеспечивает анализ типа HTTP-запроса и вызов соответствующего метода. В спецификации HTTP 1.1 предусмотрено семь типов запросов, для каждого из которых в классе `HttpServlet` предусмотрен свой метод: `doDelete()`, `doGet()`, `doHead()`, `doOptions()`, `doPost()`, `doPut()`, `doTrace()`. При реализации сервлета на базе класса `HttpServlet` нужно переопределить методы, соответствующие тем типам HTTP-запросов, которые будет обрабатывать данный сервлет. При этом передача объектов, представляющих запрос и ответ, выполняется не в виде объектов типов `ServletRequest` и `ServletResponse`, а в виде объектов типов `javax.servlet.http.HttpServletRequest` и `javax.servlet.http.HttpServletResponse`, соответственно. Интерфейсы `HttpServletRequest` и `HttpServletResponse` расширяют, соответственно, интерфейсы `ServletRequest` и `ServletResponse`, добавляя методы, которые позволяют использовать возможности протокола HTTP.

Поточные модели сервлетов.

Спецификация сервлетов предусматривает несколько моделей выполнения, влияющих на возможность параллельной обработки запросов. По умолчанию используется мультипоточная модель, в которой все запросы, относящиеся к одному сервлету, обрабатываются одним его экземпляром. При этом его метод `service()` вызывается веб-контейнером в разных потоках выполнения, каждый из которых порождается для обработки одного запроса. В этой модели все поля класса сервлета совместно используются несколькими потоками, что требует синхронизации доступа к ним во избежание непредсказуемых результатов.

Предусмотрена также однопоточная модель, когда может поддерживаться пул экземпляров сервлета, с каждым из которых работает только один поток выполнения. Пул экземпляров поддерживать необязательно, и если он не поддерживается, запросы фактически обрабатываются последовательно один за другим. Для включения однопоточной модели класс сервлета должен реализовать маркерный[1] интерфейс `javax.servlet.SingleThreadModel`.

---

[1] Интерфейс, не содержащий методов, который класс реализует для того, чтобы сообщить возможным клиентам (в рассматриваемом случае – веб-контейнеру) о своих свойствах.

Еще один случай, предусмотренный спецификацией – это строго последовательная обработка запросов. Для этого достаточно при реализации метода `service()` пометить его ключевым словом `synchronized`.

Основной моделью, рекомендуемой к использованию, является мультипоточная модель. Остальные модели не рекомендуются к использованию и их поддержка может быть прекращена в будущих версиях.

Конфигурация сервлета – интерфейс javax.servlet.ServletConfig

`getInitParameterNames()` – возвращает список имен параметров инициализации сервлета в виде объекта типа `java.util.Enumeration`.

`getInitParameter()` – возвращает значение параметра инициализации по имени в виде строки.

`getServletContext()` – позволяет получить контекст сервлета (объект типа `javax.servlet.ServletContext`), который является важной частью объектной инфраструктуры сервлетов.

Контекст сервлета – интерфейс javax.servlet.ServletContext

Объект этого типа автоматически создается веб-контейнером для каждого установленного в нем веб-приложения, то есть для каждого веб-приложения существует свой экземпляр объекта типа `ServletContext`, доступный всем его веб-компонентам.

Назначение объекта `ServletContext` довольно широкое и содержащиеся в нем методы могут быть сгруппированы по функциям:

1) доступ к параметрам конфигурации (параметрам контекста), которые указываются в дескрипторе установки. Эти параметры могут быть использованы всеми компонентами веб-приложения. Для этого предназначены следующие методы:
   - `getInitParameterNames()` – возвращает список имен параметров контекста;
   - `getInitParameter()` – возвращает значение параметра по имени;
2) обмен информацией между веб-компонентами с помощью атрибутов контекста – объектов произвольного типа, хранящихся в контексте и идентифицируемых по имени. `ServletContext` – это сравнительно долгоживущий объект, создаваемый, как правило, при запуске сервера приложений, и уничтожаемый при его останове. Соответствующее время «живут» и атрибуты контекста. Для работы с атрибутами используются методы:
   - `getAttributeNames()` – возвращает список имен атрибутов;
   - `getAttribute()` – возвращает значение атрибута по имени в виде объекта (`java.lang.Object`);
   - `setAttribute()` – добавление или изменение атрибута, имя и значение которого указываются параметрами метода, если атрибут с указанным именем уже существует, то изменяется его значение, если значение для атрибута указано как `null`, то это означает удаление атрибута;
   - `removeAttribute()` – удаление атрибута по имени;
3) регистрация сообщений и возникающих исключений в системном журнале (логе) сервера приложений с помощью метода `log();`
4) перенаправление запросов;
5) получение информации о версии спецификации сервлетов, поддерживаемой сервером приложений. Для этого предназначены методы `getMajorVersion()`

– возвращает старший номер версии (для текущей версии 2), `getMinorVersion()` – возвращает младший номер версии (для текущей версии 5).

Запрос: интерфейсы javax.servlet.ServletRequest, javax.servlet.http.HttpServletRequest

Основная модель взаимодействия клиента с сервером – это модель запрос-ответ. При этом стандартным протоколом взаимодействия является протокол HTTP, хотя в технологии сервлетов возможно расширение для обеспечения поддержки других протоколов. Поэтому для представления запросов предусмотрены два интерфейса. Первый – `javax.servlet.ServletRequest` – является базовым для любых запросов и не несет в себе никакой протокольной специфики. Второй – `javax.servlet.http.HttpServletRequest` – расширяет первый и используется для представления именно HTTP-запросов. Для каждого входящего HTTP-запроса сервер приложений создает объект этого типа.

Рассмотрим основные функции, предоставляемые объектом запроса.

1) Получение параметров запроса. Вне зависимости от используемого протокола у запроса могут быть те или иные параметры, указываемые клиентом при формировании запроса. Значения параметров представляются в виде строк. Если присутствует несколько параметров с одинаковым именем, то считается, что у параметра с этим именем несколько значений. Для работы с параметрами в интерфейсе `ServletRequest` предусмотрены следующие методы:

- `getParameterNames()` – возвращает список имен параметров, присутствующих в запросе;
- `getParameter()` – возвращает первое значение параметра с указанным именем;
- `getParameterValues()` – возвращает полный список значений параметра в виде массива строк (метод полезен при обработке форм, содержащих таблицу с флажками или список, допускающий множественный выбор).

2) Хранение атрибутов. Аналогично контексту сервлета объект запроса может выступать в роли контейнера для других объектов – атрибутов. Время жизни атрибутов запроса ограничено временем обработки запроса. Атрибуты запроса – основное средство передачи информации при перенаправлении запроса от одного веб-компонента другому. Для работы с атрибутами запроса используются методы `getAttributeNames()`, `getAttribute()`, `setAttribute()`, `removeAttribute()` полностью аналогичные соответствующим методам интерфейса `ServletContext`.

3) Работа с телом запроса. Получить доступ к телу запроса можно двумя способами:

- метод `getReader()` возвращает объект типа `java.io.BufferedReader`, который позволяет работать с телом запроса как с текстовым потоком ввода;
- метод `getInputStream()` возвращает объект типа `javax.servlet.ServletInputStream`, который позволяет работать с телом запроса как с бинарным потоком ввода. Это позволяет обрабатывать запросы, тело которых содержит не параметры формы, а некоторые файлы.

Размер тела запроса можно получить методом `getContentLength()`, а тип содержимого запроса – методом `getContentType()`. Метод `getCharacterEncoding()` возвращает название кодировки, в которой представлено тело запроса.

4) Получение информации о клиенте. С запросом может быть передана информация о региональных настройках клиента, так называемая локаль (locale). Извлечь ее можно методом `getLocale()` в виде объекта класса `java.util.Locale`. Узнать, с какого узла сети

пришел запрос можно с помощью двух методов: `getRemoteHost()` возвращает доменное имя узла, `getRemoteAddr()` возвращает его IP-адрес.

Все рассмотренные методы предусмотрены в базовом интерфейсе `ServletRequest`. В интерфейсе `HttpServletRequest` дополнительно определены следующие функции.

1) Метод `getMethod()` возвращает имя метода, которым был послан запрос.

2) В заголовочной части HTTP-запроса присутствуют так называемые заголовки (пары имя/значение). Для работы с ними предусмотрены следующие методы:

- `getHeaderNames()` – возвращает список имен заголовков;
- `getHeader()` – возвращает в виде строки первое значение заголовка по имени;
- `getHeaders()` – возвращает список строковых значений заголовка по имени.
- `getIntHeader()` – возвращает первое целочисленное (типа `int`) значение заголовка по имени;
- `getDateHeader()` – возвращает первое значение заголовка типа даты (`java.util.Date`) по имени.

3) С HTTP-запросом могут быть связаны куки (cookies) – информация, которая ранее была получена клиентом с HTTP-ответом от этого веб-узла и сохранена на нем для последующей передачи этому же веб-узлу при повторном обращении. Механизм куки позволяет серверному ПО сохранять временную информацию на клиентских машинах и позже получать ее вместе с запросом. За счет этого механизма можно реализовать поддержку сессий, не предусмотренную на уровне протокола HTTP. Извлечь куки из запроса можно методом `getCookies()` в виде массива объектов типа `javax.servlet.http.Cookie`. Каждый объект куки содержит имя, некоторое значение, момент времени устаревания (после которого эта информация на клиенте будет удалена) и некоторые дополнительные атрибуты.

4) Метод `getSession()` позволяет получить доступ к HTTP-сессии, которая представляется для каждого отдельного сеанса каждого пользователя в виде объекта типа `javax.servlet.http.HttpSession`. Подробнее о сессиях см. далее.

5) Из объекта запроса можно извлечь информацию о том, по какому URL было выполнено обращение клиентом. Напомним, что в общем случае URL имеет вид `protocol://hostname:port/pathname?search#hash`. В веб-приложениях на платформе Java EE компонент `pathname`, по которому клиент может обратиться к веб-компоненту, конструируется последовательно из трех частей:

/<корень веб-приложения>/<путь к сервлету>/<информация о пути>

Корень веб-приложения (context path) – это уникальный в пределах сервера приложений идентификатор веб-приложения, который задается в установочном дескрипторе в виде виртуального пути `/имя/имя/…`. После установки приложения данный путь используется как часть URL при обращении к компонентам приложения.

Путь к сервлету (servlet path) также задается в установочном дескрипторе и должен быть уникальным в пределах веб-модуля. С одним сервлетом может быть связано несколько разных путей.

Если путь к сервлету содержит `/*` как последний путевой элемент, то при обращении к нему URL в этой части может содержать произвольную конструкцию вида `/имя/имя/имя`., которая представляет собой информацию о пути (path info).

Для извлечения компонентов пути из URL запроса предусмотрены соответствующие методы: `getContextPath()`, `getServletPath()`, `getPathInfo()`. Кроме того, существуют методы для получения всего URL запроса или отдельных его частей.

Ответ: интерфейсы javax.servlet.ServletResponse и javax.servlet.http.HttpServletResponse.

Для ответов, так же как и для запросов, выполнена декомпозиция API: независящая от протокола часть вынесена в базовый интерфейс `javax.servlet.ServletResponse`; а часть, специфичная для протокола HTTP, сосредоточена в производном интерфейсе `javax.servlet.http.HttpServletResponse`.

1) Управление генерацией ответа. Ответ может быть в одном из двух состояний: переданный (committed) и не переданный (not committed). Ответ считается переданным, если хотя бы один байт был отправлен клиенту, до этого момента ответ находится в не переданном состоянии. При этом вывод, генерируемый веб-компонентом, на практике всегда буферизуется для повышения производительности. Это означает, что ответ переходит в переданное состояние не с началом генерации вывода, а лишь после первого переполнения буфера. Для управления буфером предусмотрены методы `getBufferSize()`, возвращающий текущий размер буфера в байтах (`0`, если буферизация отключена); и `setBufferSize()`, увеличивающий размер буфера. Изменить размер буфера можно только до начала генерации ответа, иначе используется значение по умолчанию 8 КБ. Метод `resetBuffer()` позволяет очистить буфер с сохранением статуса ответа и заголовков, установленных ранее; а метод `reset()` – очистить буфер, уничтожить заголовки и сбросить статус ответа. Reset-методы можно использовать только в не переданном состоянии, иначе возникает исключение `java.lang.IllegalStateException`. Метод `flushBuffer()` передает содержимое буфера клиенту, при этом ответ переходит в переданное состояние. Узнать состояние ответа можно методом `isCommitted()`.

2) Формирование тела ответа. Тело ответа может содержать текстовые (только текст, HTML-документ, XML-документ) или бинарные (рисунки, видео, звук) данные. Вывод данных разных типов выполняется разными средствами, получить которые можно из объекта ответа. Сам по себе этот объект сгенерировать вывод не позволяет.

Для генерации текстовых ответов используется объект типа `java.io.PrintWriter`, получить который из ответа можно методом `getWriter()`. Для бинарных ответов используют объект типа `javax.servlet.ServletOutputStream`, получить который можно методом `getOutputStream()`. При формировании ответа можно использовать только один из объектов: либо `PrintWriter`, либо `ServletOutputStream`.

3) Определение характеристик ответа. Установить MIME-тип генерируемого ответа нужно еще до получения объектов `PrintWriter` или `ServletOutputStream` методом `setContentType()`. Как часть идентификатора MIME-типа для текстовых данных можно указать и кодировку. Например,

```
response.setContentType( "text/html; charset=windows-1251" );
```

Для работы с кодировкой и локалью ответа используются методы `getCharacterEncoding()`, `setCharacterEncoding()`, `getLocale()`, `setLocale()`.

Все рассмотренные методы присутствуют в интерфейсе `ServletResponse`; интерфейс `HttpServletResponse` добавляет следующие методы.

1) Методы для работы с заголовками HTTP-ответа:

- `setHeader()` – создает в ответе заголовок с указанным именем и значением. Если заголовок с таким именем существует, его значение перезаписывается. Методы `setIntHeader()` и `setDateHeader()` выполняют аналогичные действия, но работают со значением заголовка типа `int` и `java.util.Date`, соответственно;
- `addHeader()` – добавляет еще одно значение к существующему заголовку с указанным именем. Если заголовок не существует, то он создается. Для значений заголовков типа `int` и `java.util.Date` аналогичные действия выполняют методы `addIntHeader()` и `addDateHeader()`;

- `containsHeader()` – проверяет существование заголовка с указанным именем в ответе.

2) Спецификация HTTP предусматривает числовой статус ответа, по которому клиент (браузер) может судить о возникших при обработке запроса ошибках. Для каждого кода статуса в интерфейсе `HttpServletResponse` определена константа (например, `SC_OK` для кода 200 и т.д.). Установить статус ответа можно методом `setStatus()`.

3) Метод `addCookie()` позволяет добавить куки к ответу.

4) Методы для перенаправления запроса (см. ниже).

HTTP-сессия: интерфейс javax.servlet.http.HttpSession.

Изначально протокол HTTP не содержал никакого механизма для поддержания сеанса работы клиента с веб-сервером, но в рамках сервера веб-приложений такие механизмы были обеспечены. Важность сеанса состоит в том, что часто нужно хранить некоторую специфичную для конкретного пользователя информацию в промежутках между его обращениями к серверу, причем сохранять ее нужно на сервере.

Для каждого сеанса пользователя веб-контейнер генерирует уникальный идентификатор и создает объект типа `javax.servlet.http.HttpSession`, в котором хранится связанная с сеансом информация.

На платформе Java EE предусмотрены три механизма для поддержания сеанса по протоколу HTTP: на основе куки, на основе URL (URL rewriting) и на основе средств, предусмотренных протоколом SSL[2] (Secure Socket Layer).

Механизм на основе куки используется по умолчанию. При этом идентификатор сеанса (сессии) прикрепляется к ответу в виде куки с именем `JSESSIONID`. При последующих обращениях, сделанных в рамках одного сеанса, браузер прикрепляет данный куки к запросу. Данный механизм работает только в случае, если поддержка куки в браузере не отключена.

Механизм на основе URL заключается в том, что идентификатор сессии прикрепляется к URL в виде GET-параметра с именем `jsessionid`. Этот механизм работает, даже если отключены куки. Однако при этом необходимо внедрять соответствующий GET-параметр во все ссылки, ведущие к тому же приложению во всех генерируемых им страницах-ответах. Для включения идентификатора сессии в состав URL предусмотрен метод `HttpServletResponse.encodeURL()`. С помощью этого метода должны обрабатываться все ссылки, которые присутствуют на всех генерируемых страницах-ответах.

Протокол SSL предусматривает встроенный механизм для поддержания сессии. Он может использоваться совместно с протоколом HTTP в случае взаимодействия с веб-контейнером по зашифрованному каналу.

Если запрос не относится ни к одному из существующих сеансов, то начинается новый сеанс, при этом генерируется идентификатор сессии и создается объект типа `HttpSession`, в котором хранится идентификатор сессии на сервере, и который обычно прикрепляется к ответу в виде куки. Веб-контейнер, извлекая идентификатор сессии из запроса (из куки или из GET-параметра), может определить, к какому сеансу относится этот запрос. Если в браузере отключены куки, а веб-приложение не выполняет обработку URL методом `encodeURL()`, то каждый запрос такого клиента приводит к созданию нового сеанса.

Слабая связанность веб-клиента и веб-контейнера требует применения тайм-аута как единственного надежного средства для определения момента окончания сеанса работы пользователя с веб-приложением. По умолчанию принимается тайм-аут в 30 минут. Если в течение тайм-аута не было обращений от пользователя, то сеанс этого пользователя

[2] Протокол сеансового уровня, который можно использовать совместно с HTTP.

считается завершенным. Соответствующий объект сессии уничтожается, а идентификатор сессии становится недействительным.

Получить объект сессии можно методом `getSession()` интерфейса `HttpServletRequest`. В интерфейсе `HttpServletRequest` также есть методы, позволяющие узнать содержащийся в запросе идентификатор сессии[3] (метод `getRequestSessionId()`) и способ его передачи – вместе с куки (метод `isRequestedSessionIdFromCookie()`) или как GET-параметр в составе URL (метод `isRequestedSessionIdFromURL()`). Метод `isRequestedSessionIdValid()` позволяет проверить, действительна ли сессия, идентификатор которой был передан с запросом.

Интерфейс `javax.servlet.http.HttpSession` содержит методы, позволяющие узнать все параметры сессии и работать с атрибутами сессии. Метод `getId()` возвращает идентификатор сессии, `getCreationTime()` – момент создания сессии, `getLastAccessTime()` – момент последнего обращения пользователя в рамках этого сеанса, `getMaxInactiveInterval()` – тайм-аут сессии, `isNew()` проверяет, была ли создана сессия при поступлении данного запроса или нет. Метод `setMaxInactiveInterval()` позволяет установить величину тайм-аута.

Объект сессии, наряду с контекстом сервлета и запросом, может выступать в роли контейнера для других объектов – атрибутов. Время жизни атрибутов сессии ограничено временем сеанса работы пользователя с веб-приложением. Границы этого сеанса определяются так, как описано выше. Атрибуты сессии – это основной механизм для сохранении информации на сервере между обращениями одного и того же пользователя в рамках одного сеанса работы с веб-приложением. Например, в виде атрибутов сессии сохраняют ссылки на сессионные компоненты с состоянием.

На платформе Java EE веб-сессия локальна для веб-модуля, то есть при обращении пользователя к разным веб-модулям будет создано два разных сеанса, даже если эти модули были установлены в составе одного Java EE-приложения.

Перенаправление запросов

При разработке сложных веб-приложений, содержащих несколько веб-компонентов, участвующих в обработке запросов, часто возникает необходимость организовать взаимодействие между ними в рамках процедуры обработки одного запроса. Делается это путем перенаправления запроса, поступившего к одному из веб-компонентов, на обработку другому веб-компоненту. Например, запрос на получение списка продукции от пользователя, не прошедшего аутентификацию, может быть перенаправлен на страницу ввода имени пользователя и пароля.

Перенаправление может выполняться двумя разными способами. Первый основан на механизме, являющемся частью протокола HTTP. Его суть в том, что клиенту генерируется специальный перенаправляющий ответ, который содержит не запрошенные данные, а URL, по которому клиенту необходимо обратиться для их получения. При получении перенаправляющего ответа клиент (браузер) генерирует новый запрос по полученному URL. Генерация перенаправляющего URL выполняется методом `sendRedirect()` интерфейса `javax.servlet.http.HttpServletResponse`. В качестве параметра указывается URL того веб-компонента, на который следует перенаправить клиента (этот URL может быть указан как относительный). Специфика данного способа в том, что после генерации перенаправляющего ответа обработка исходного запроса считается законченной и информация о нем теряется. При использовании метода `sendRedirect()` существуют ограничения:

1) вызывать его для не переданного ответа. Если ответ был передан клиенту, то вызов приводит к выбросу исключения `java.lang.IllegalStateException`;

[3] К моменту получения запроса содержащийся в нем идентификатор сессии может быть недействительным.

2) если в момент вызова метода буфер вывода был не пустой, он очищается;

3) после вызова этого метода запрос считается обработанным, ответ переходит в переданное состояние, а весь вывод, генерируемый далее, игнорируется.

Второй способ перенаправления заключается в использовании диспетчера запросов (объекта типа `javax.servlet.RequestDispatcher`), который можно получить методами `getRequestDispatcher()` и `getNamedDispatcher()` интерфейса `javax.servlet.ServletContext`. Первый метод в качестве параметра требует относительный URL[4] веб-компонента, на который будет сделано перенаправление. Относительный путь отсчитывается от корня веб-приложения, указанного в установочном дескрипторе. Второй метод в качестве параметра требует имя веб-компонента в текущем веб-приложении, на который будет сделано перенаправление.

При необходимости перенаправить запрос веб-компоненту другого веб-приложения необходимо сначала получить из контекста своего приложения (объект типа `ServletContext`) методом `getContext()` контекст целевого приложения, а уже из него получить диспетчер запросов. В качестве параметра метод `getContext()` принимает в виде строки корень целевого веб-приложения.

В диспетчере запросов перенаправление может быть сделано одним из двух методов: `forward()` или `include()`. В обоих случаях в качестве параметров передаются запрос (`ServletRequest`) и ответ (`ServletResponse`).

Метод `forward()` передает запрос на обработку другому компоненту, который и должен сгенерировать ответ. Ограничения метода `forward()` аналогичны тем, что имеют место для метода `ServletResponse.sendRedirect()`.

Метод `include()` позволяет включить ответ, генерируемый другим компонентом, в состав ответа, генерируемого данным компонентом. Это используется для блочного конструирования ответа на этапе выполнения. Ограничения для метода `include()` запрещают вызываемому компоненту изменять заголовки ответа. Метод `include()` может быть вызван в любой момент обработки запроса вне зависимости от состояния ответа (передан/не передан).

В интерфейсе `HttpServletResponse` существует метод `sendError()`, который позволяет послать в качестве ответа клиенту стандартное сообщение об ошибке, оформленное специфичным для данной реализации сервера приложений образом с указанием кода ошибки (статуса HTTP-ответа) и сообщения об ошибке. Ограничения для метода `sendError()` такие же, как и для метода `sendRedirect()`.

Пример сервлета

Сервлет `BookServlet` возвращает HTML-страницу с подробными сведениями о книге, идентификатор которой передается в качестве параметра `bookId`. Для выбора нужного действия сервлета используется параметр `action`.

```
package J2EESample.Servlet;

import java.io.*;
import java.sql.*;
import javax.naming.*;
import javax.servlet.*;
import javax.sql.*;

public class BookServlet extends GenericServlet {
   public void service(
      ServletRequest request,
      ServletResponse response
```

[4] Относительный URL начинается с `/`, например, `/addProduct.jsp`.

```
   ) throws IOException, ServletException {
      String action = request.getParameter("action");
      if ("info".equals(action)) {
         String bookIdStr = request.getParameter("bookId");
         int bookId = 0;
         if (bookIdStr != null)
            bookId = Integer.parseInt(bookIdStr);
         Book book = new BookDAO().getBook(bookId);

         response.setContentType("text/html");
         PrintWriter responseWriter = response.getWriter();
         responseWriter.println("<html>");
         responseWriter.println("<head>");
         responseWriter.println("<title>Book Info</title>");
         responseWriter.println("</head>");
         responseWriter.println("<body>");
         if (book != null) {
            responseWriter.println("<h3>" + book.getTitle() + "</h3>");
            responseWriter.println("Author: " + book.getAuthor() + "</p>");
            ...
         } else
            responseWriter.println("<h3>Book Not Found</h3>");
         responseWriter.println("</body>");
         responseWriter.println("</html>");
         responseWriter.close();
      }
   }
}
```

Так как сервлеты могут использоваться в двух качествах – контроллера и presentation component, то приведенный пример можно реорганизовать таким образом, чтобы не смешивались функции контроллера и представления. Для этого часть кода, предназначенную для генерации результирующей HTML-страницы, вынесем в отдельный сервлет `BookInfoServlet`, а в исходном сервлете `BookServlet` будем выполнять перенаправление запроса. Связь между данными веб-компонентами выполняется через атрибут запроса `book`, в котором хранятся сведения о книге в виде объекта класса `Book`.

```
public class BookServlet extends GenericServlet {
   public void service(
      ServletRequest request,
      ServletResponse response
   ) throws IOException, ServletException {
      String action = request.getParameter("action");
      if ("info".equals(action)) {
         String bookIdStr = request.getParameter("bookId");
         int bookId = 0;
         if (bookIdStr != null)
            bookId = Integer.parseInt(bookIdStr);
         Book book = new BookDAO().getBook(bookId);
         request.setAttribute("book", book);
         RequestDispatcher dispatcher = getServletConfig()
            .getServletContext()
            .getNamedDispatcher("BookInfoServlet");
         dispatcher.forward(request, response);
      }
   }
}

public class BookInfoServlet extends GenericServlet {
   public void service(
      ServletRequest request,
      ServletResponse response
   ) throws IOException, ServletException {
```

```
        Book book = request.getAttribute("book");

        response.setContentType("text/html");
        PrintWriter responseWriter = response.getWriter();
        responseWriter.println("<html>");
        responseWriter.println("<head>");
        responseWriter.println("<title>Book Info</title>");
        responseWriter.println("</head>");
        responseWriter.println("<body>");
        if (book != null) {
           responseWriter.println("<h3>" + book.getTitle() + "</h3>");
           responseWriter.println("Author: " + book.getAuthor() + "</p>");
           ...
        } else
           responseWriter.println("<h3>Book Not Found</h3>");
        responseWriter.println("</body>");
        responseWriter.println("</html>");
        responseWriter.close();
    }
}
```