1) Написать функцию ява скрипт, с помощью которой можно закрасить все текстовые поля в желтый цвет.

```
<html>
<head>
<script>
function dofill()
{
  for (var i=0; i<document.getElementsByName('text').length; i++)
      document.getElementsByTagName('input').style.backgroundColor="yellow";
}
</script>
</head>
<body>
   <input type="text"  name="text">     <input type="text" name="text">     <input type="text" name="text">
   <a href="javascript:dofill()">re</a>
</body>
</html>
```

ИЛИ

```
function dofill(){
  for (var i=0; i<document.getElementsByTagName('input').length; i++)
      document.getElementsByTagName('input').style.backgroundColor="yellow";
```

2) Написать, jsp страницу, передающую параметр "name"

```
<html>
<head></head>
<body>
<jsp:forward page="/test">
   <jsp:param name="level" value="<%=3%>" />
</jsp:forward>
</body>
</html>
```

ИЛИ

```
<html>
<head></head>
<body>
<% response.sendRedirect("test?level=3"+); %>
</body>
</html>
```

3) Код сервлета, возв. на все запросы код сост. 404. В одном нужно было написать сервлет, возвращающий 404.

```
HttpServletResponse protected void
            doGet( HttpServletRequest req, HttpServletResponse res)
            throws ServletException{
            // тело основного метода,
            //отсутствие запрашиваемого ресурса
            res.sendError(404); }
```

4) написать джаваскрипт функцию, открывающую в новом окне сайт гугл

```
function openWin() { myWin= open("http://www.google.ru"); }
```

ИЛИ

```
function new_window()
{
window.open("http://www.google.ru ",'myWin','top=15, left=20, menubar=0, toolbar=0, location=0, directories=0, status=0, scrollbars=0,
resizable=0, width=400, height=300');
}
```

5) написать конфигурацию сервлета, которая бы делала перенавправление всех запросов к html и xhtml страницам на наш сервлет

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
   <servlet>
      <servlet-name>myserv</servlet-name>
      <servlet-class>sample.MyServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>myserv</servlet-name>
      <url-pattern>/*</url-pattern>
   </servlet-mapping>
</web-app>
```

6) Написать html-страницу - несложная. Два текстовых поля. один комбобокс, проверка значений на джаваскрипт, отправка из формы на сервер.

index.html:

```
<html>
    <head>
        <title>title</title>
        <script src="sc.js" type="text/javascript"></script>
    </head>
    <body>
        <form method="POST">
            <input type="text" id="text_1" onchange="checkTextField('text_1','submitButton')">
            <input type="text" id="text_2">
            <select>
              <option>one</option>
              <option>two</option>
              <option>three</option>
            <select>
            <input type="submit" id="submitButton">
        </form>
    </body>
</html>
```

sc.js:

```
function checkTextField(textFieldId, buttonId){
   var textValue = document.getElementById(textFieldId).value;
   var y = parseFloat(textValue);
   if (isNaN(y)) {
      document.getElementById(buttonId).disabled=true;
   } else {
      document.getElementById(buttonId).disabled=false;
   }
}
```

7) Код JSP-страницы, показывающий значение параметра "name" из HTTP-запроса.

```
<html>
<head></head>
<% String name = request.getParameter("name");%>
<body><%= name %></body>
</html>
```

8) Конфигурация сервлета (класс com#sample.MyServlet), отправ. ему на обработку все запросы на html и xhtml-стр.

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
   <servlet>
      <servlet-name>myserv</servlet-name>
      <servlet-class>sample.MyServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>myserv</servlet-name>
      <url-pattern>/*</url-pattern>
   </servlet-mapping>
</web-app>
```

Описание DOM

Объектная модель документа (DOM) стандарт, регламентирующий способ представления содержимого документа (в частности веб-страницы) в виде набора объектов. Под содержимым понимается все, что может находиться на веб-странице: рисунки, ссылки, абзацы, текст и т. д.

DOM — это платформенно-независимый интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML-документов.

Стандартизирована W3C.

Документ в DOM представляет собой дерево узлов.

Узлы связаны между собой отношением «родитель-потомок».

Используется для динамического изменения страниц HTML

DOM разбит на три уровня. Первый уровень является первой версией стандарта и пока что единственной законченной. Он состоит из двух разделов: первый является ядром и определяет принципы манипуляции со структурой документа (генерация и навигация), а второй посвящен представлению в DOM элементов HTML, определяемых одноименными тегами.

Второй и третий уровни описывают модель событий, дополняют таблицы стилей, проходы по структуре.

Управление сессиями (HTTP session).

Основу средств создания сеанса связи с клиентом посредством сессии в Java составляет интерфейс HttpSession. Для каждого клиента сервлет/JSP может создать объект HttpSession, который ассоциируется только с этим клиентом и может быть доступен только этому конкретному клиенту. Объект HttpSession действует как Hashtable (хэш-таблица), где можно хранить любое число пар ключ/объект. Объект HttpSession доступен другим сервлетам в том же приложении. Чтобы извлечь сохраненный ранее объект, необходимо только передать ключ.

Получение ссылки на объект HttpSession осуществляется за счет метода getSession у объекта-запроса (например, HttpServletRequest):
public HttpSession getSession () - метод возвращает объект типа HttpSession, если он существует, иначе создает и возвращает новый.

getAttribute(), getAttributeNames, setAttribute(), removeAttribute(). Эти методы используются для установки, получения и удаления объектов из сессии пользователя.
getId(). Каждая сессия, созданная сервером, имеет уникальный id, ассоциированный с ней для идентификации сессии среди других сессий. Метод как раз возвращает такой id.
getCreationTime(). Возвращает long значение, определяющее дату и время создания данной сессии.
getLastTimeAccess(). Возвращает значение long, обозначающее последний визит на сайте.
isNew(). Возвращает boolean, определяя новая ли сессия. Это значит, что если это первая страница на сайте, на которую нажал пользователь, то она или только что была создана сессия для пользователя, то метод возвращает true.
invalidate(). Аннулирует сессию. Этот метод можно использовать на странице 'logout', позволяя пользователю закончить сессию. Если после этого пользователь снова зайдет на сайт - создастся новая сессия под него.

Css источники, правила, приоритеты.
CSS – каскадные таблицы стилей – язык описания внешнего вида документа, написанного с использование языка разметки. Чаще всего используется для оформления веб-страниц в формате HTML и XHTML, но может применяться к XML -документам. CSS используется для задания штифтов, цветов, расположения и др. аспектов представления документа. Основная цель – разделение содержимого (написанного на HTML или другом языке разметки) и представления документа (написанного на CSS). Разделение способствует возможности увеличения доступности документа, уменьшения сложности и повторяемости в структурном содержимом, а также предоставить большую гибкость и возможность управления его представлением.
<LINK REL=STYLESHEET TYPE="text/css" HREF="styles.css">
Источники:
Авторские стили (информация стилей, предоставляемая автором страницы) в виде: Inline-стилей — стиль элемента указывается в его атрибуте style. Встроенных стилей — блоков CSS внутри самого HTML-документа.Внешних таблиц стилей — отдельного файла .css.
Пользовательские стили:
Локальный CSS-файл, указанный пользователем в настройках браузера, переопределяющий авторские стили.
Стиль браузера:
Стандартный стиль, используемый браузером по умолчанию для представления элементов.

Правила:
Таблица стилей состоит из набора правил. Каждое правило состоит из набора селекторов и блока определений:
```
селектор, селектор {
  свойство: значение;
  свойство: значение;
  свойство: значение;
}
```
Пример:
```
div, td {
  background-color: red;
}
```

Приоритеты
Если к одному элементу «подходит» сразу несколько стилей, применён будет наиболее приоритетный.
Приоритеты рассчитываются таким образом (от большего к меньшему):
1. свойство задано при помощи !important;
2. стиль прописан напрямую в теге;
3. количество идентификаторов (#id) в селекторе (чем больше, тем больше приоритет);
4. количество классов (.class) и псевдоклассов (:pseudoclass) в селекторе;
5. количество имён тегов в селекторе.
Имеет значение относительный порядок расположения свойств — свойство, указанное позже, имеет приоритет

Методы HTTP
Метод HTTP— последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами.
OPTIONS — Используется для определения возможностей веб-сервера или параметров соединения для конкретного ресурса.
GET — запрос содержимого ресурса. С помощью метода GET можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса. Запросы типа GET считаются идемпотентными — многократное повторение одного и того же запроса GET должно приводить к одинаковым результатам (при условии, что сам ресурс не изменился за время между запросами). Это позволяет кэшировать ответы на запросы GET.
POST — передача данных ресурсу. В отличие от метода GET, метод POST не считается идемпотентным, то есть многократное повторение одних и тех же запросов POST может возвращать разные результаты (например, после каждой отправки комментария будет появляться одна копия этого комментария).
HEAD — аналог GET, но в ответе отсутствует тело. Запрос HEAD обычно применяется для извлечения метаданных, проверки наличия ресурса (валидация URL) и чтобы узнать, не изменился ли он с момента последнего обращения. Заголовки ответа могут кэшироваться. При несовпадении метаданных ресурса с соответствующей информацией в кэше копия ресурса помечается как устаревшая.
PUT — загрузка содержимого запроса на указанный URI.
Сообщения ответов сервера на метод PUT не кэшируются.
PATCH — Аналогично PUT, но применяется только к фрагменту ресурса.
DELETE — Удаляет указанный ресурс.
TRACE — Возвращает полученный запрос так, что клиент может увидеть, какую информацию промежуточные сервера добавляют или изменяют в запросе.
LINK — Устанавливает связь указанного ресурса с другими.
UNLINK — Убирает связь указанного ресурса с другими.
CONNECT — Преобразует соединение запроса в прозрачный TCP/IP туннель, обычно чтобы содействовать установлению защищенному SSL соединению через не шифрованный прокси.

Конфигурация сервлета, файл web.XML
Дескриптор веб-приложения описывает классы, ресурсы и конфигурацию приложения, а также их использование веб-сервером для получения веб-запросов. При получении веб-сервером запроса для приложения он использует дескриптор развертывания, чтобы сопоставить URL запроса с кодом, который должен обработать запрос.
Дескриптор развертывания – это файл с названием web.xml. Он находится в WAR приложения в каталоге WEB-INF/. Это XML-файл, корневой элемент которого – <web-app>.
Ниже приведен простой пример web.xml, сопоставляющий все пути URL (/*) с классом сервлета mysite.server.ComingSoonServlet:

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
   <servlet>
      <servlet-name>comingsoon</servlet-name>
      <servlet-class>mysite.server.ComingSoonServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>comingsoon</servlet-name>
      <url-pattern>/*</url-pattern>
   </servlet-mapping>
</web-app>
```

web.xml определяет сопоставления между путями URL и сервлетами, обрабатывающими запросы с этими путями. Веб-сервер использует эту конфигурацию для определения сервлета, который должен обрабатывать определенный запрос, и вызова метода класса, соответствующего методу запроса (например, метод doGet() для запросов HTTP GET).
Чтобы сопоставить URL сервлету, необходимо объявить сервлет с элементом <servlet>, а затем определить сопоставление пути URL к объявлению сервлета с элементом <servlet-mapping>.

Структура HTML-документа.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
   <HEAD>
   <TITLE>Заголовок документа</TITLE>
   </HEAD>
<BODY>
Текст документа
</BODY>
</HTML>
```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"> –- обозначает тип документа и формат. Облегчает распознавание другими программами этого файла. Указывает в начале версию стандарта HTML и язык документа. Рекомендуется оставлять без изменений для одинакового отображения в большинстве браузеров.
HTML-документ начинается с тега <html>, который сообщает браузеру о начале документа HTML и заканчивается тегом </html>, который информирует браузер о достижении конца документа HTML. После тега </html> больше ничего не пишут.
Текст между тегами <head> и </head> является информацией заголовка документа. Эта информация не выводится в окне браузера.
Текст «Это заголовок страницы» между тегами <title> и </title> является заголовком документа. Этот заголовок выводится в строке заголовка окна браузера.
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251"> — укзаывает на то, что документ содержит HTML-текст в кодировке windows-1251 — это кириллица.
<link href="css/styles.css" rel="stylesheet" type="text/css"> — подключает к документу таблицу стилей CSS.
Текст между тегами <body> и </body> является текстом, который будет выведен в окне браузера. Текст «Здравствуйте!» между тегами <h1> и </h1> будет отображен стилем заголовка, обычно жирным шрифтом большего размера.
<!-- Это комментарий --> — комментарии в HTML заключаются в <!-- и -->
Тег означает, что начинается новый параграф, тег означает конец параграфа.
Текст «Этот текст выводится жирным шрифтом.» между тегами и будет выведен жирным шрифтом.

Правила создания html-форм
Предназначены для обмена данными междупользователем и сервером.
Документ может содержать любое число форм, ноодновременно на сервер может быть отправлена толькоодна из них.
Вложенные формы запрещены.
Границы формы задаются тегами <form>...</form>.
<form name="forma zakaza" method="post" action="obrabotchik.php"> <!-- сюда вставляют различные элементы --> </form>
Атрибуты:
NAME – определяет имя формы, уникальное для данного документа. Используется только , если в документе присутствует несколько форм.
ACTION – обязательный атрибут. Указывает путь к скрипту(или программе) сервера, обслуживающему данную форму.
METHOD – определяет способ отправки содержимого html формы. Возможные значения GET (по умолчанию) и POST.
Пример HTML-формы

```
<form method="POST" action="handler.php">
<p><b>Как по вашему мнению расшифровывается
аббревиатура &amp;quot;ОС&amp;quot;?</b></p>
<p><input type="radio" name="answer"
value="a1">Офицерский состав<Br>
<input type="radio" name="answer"
value="a2">Операционная система<br>
<input type="radio" name="answer"
value="a3">Большой полосатый мух</p>
```

<input type="submit">
</form>

+ и – Аякса
AJAX (Asynchronous Javascript and XML) —подход к построению интерактивных пользовательских интерфейсов веб-приложений.Основан на «фоновом» обмене данными браузера с веб-сервером. При обмене данными между клиентом и сервером веб-страница не перезагружается полностью.
Преимущества
* Экономия трафика
* Уменьшение нагрузки на сервер
* Ускорение реакции интерфейса
Недостатки
* Отсутствие интеграции со стандартными инструментами браузера
Динамически создаваемые страницы не регистрируются браузером в истории посещения страниц, поэтому не работает кнопка «Назад», предоставляющая пользователям возможность вернуться к просмотренным ранее страницам, но существуют скрипты, которые могут решить эту проблему.
Другой недостаток изменения содержимого страницы при постоянном URL заключается в невозможности сохранения закладки на желаемый материал. Частично решить эти проблемы можно с помощью динамического изменения идентификатора фрагмента (части URL после #), что позволяют многие браузеры.
* Динамически загружаемое содержимое недоступно поисковикам (если не проверять запрос, обычный он или XMLHttpRequest)
Поисковые машины не могут выполнять JavaScript, поэтому разработчики должны позаботиться об альтернативных способах доступа к содержимому сайта.
* Старые методы учёта статистики сайтов становятся неактуальными
* Усложнение проекта
Перераспределяется логика обработки данных — происходит выделение и частичный перенос на сторону клиента процессов первичного форматирования данных. Это усложняет контроль целостности форматов и типов. Конечный эффект технологии может быть нивелирован необоснованным ростом затрат на кодинг и управление проектом, а также риском снижения доступности сервиса для конечных пользователей.

REST и RPC
REST (сокр. англ. Representational State Transfer, «передача состояния представления») — подход к архитектуре сетевых протоколов, обеспечивающих доступ к информационным ресурсам. Самой известной системой, построенной в значительной степени по архитектуре REST, является современная Всемирная паутина.
Данные должны передаваться в виде небольшого количества стандартных форматов (например HTML, XML, JSON). Сетевой протокол (как и HTTP) должен поддерживать кэширование, не должен зависеть от сетевого слоя, не должен сохранять информацию о состоянии между парами «запрос-ответ». Утверждается, что такой подход обеспечивает масштабируемость системы и позволяет ей эволюционировать с новыми требованиями.
Антиподом REST является подход, основанный на вызове удаленных процедур (Remote Procedure Call — RPC). Подход RPC позволяет использовать небольшое количество сетевых ресурсов с большим количеством методов и сложным протоколом. При подходе REST количество методов и сложность протокола строго ограничены, из-за чего количество отдельных ресурсов должно быть большим.

Удалённый вызов процедур (или Вызов удалённых процедур) (от англ. Remote Procedure Call (RPC)) — класс технологий, позволяющих компьютерным программам вызывать функции или процедуры в другом адресном пространстве (как правило, на удалённых компьютерах). Обычно, реализация RPC технологии включает в себя два компонента: сетевой протокол для обмена в режиме клиент-сервер и язык сериализации объектов (или структур, для необъектных RPC). Различные реализации RPC имеют очень отличающуюся друг от друга архитектуру и разнятся в своих возможностях: одни реализуют архитектуру SOA,
другие CORBA или DCOM. На транспортном уровне RPC используют в основном протоколы TCP и UDP, однако, некоторые построены на основе HTTP (что нарушает архитектуру ISO/OSI, так как HTTP изначально не транспортный протокол).
дея вызова удалённых процедур (Remote Procedure Call — RPC) состоит в расширении хорошо известного и понятного механизма передачи управления и данных внутри программы, выполняющейся на одной машине, на передачу управления и данных через сеть. Средства удалённого вызова процедур предназначены для облегчения организации распределённых вычислений и создания распределенных клиент-серверных информационных систем. Наибольшая эффективность использования RPC достигается в тех приложениях, в которых существует интерактивная связь между удалёнными компонентами с небольшим временем ответов и относительно малым количеством передаваемых данных. Такие приложения называются RPC-ориентированными.

Жизненный цикл сервлета.
Жизненный циклом управляет веб-контейнер.
Методы,управляющие жизненным циклом,должен вызывать только контейнер.

1.Загрузка класса сервлета
2.Создание экземпляра
3.Вызов метода Init()
4.Вызов метода service()
5.Вызов метода destroy()

Жизненным циклом сервлета управляет веб-контейнер и только он должен вызывать методы, управляющие жизненным циклом сервлета.
Жизненный цикл сервлета управляется контейнером, в котором сервлет был развернут. Когда запрос отображается на сервлет, контейнер выполняет следующие шаги:
Если экземпляр сервлета не существует, Web-контейнер
Загружает класс сервлета.
Создает экземпляр класса сервлета.
Инициализирует экземпляр сервлета, вызывая метод init(). Этот метод инициализирует сервлет и вызывается в первую очередь, до того, как сервлет сможет обслуживать запросы. За весь жизненный цикл метод init() вызывается только однажды.

Вызывает метод service(), передавая ему объекты запроса и отклика.
Обслуживание клиентского запроса. Каждый запрос обрабатывается в своем отдельном потоке. Контейнер вызывает метод service() для каждого запроса. Этот метод определяет тип пришедшего запроса и распределяет его в соответствующий этому типу метод для обработки запроса. Разработчик сервлета должен предоставить реализацию для этих методов. Если поступил запрос, метод для которого не реализован, вызывается метод родительского класса и
В случае если контейнеру необходимо удалить сервлет, он вызывает метод destroy(), который снимает сервлет из эксплуатации. Подобно методу init(), этот метод тоже вызывается единожды за весь цикл сервлета.

DOM и BOM

DOM — это платформенно-независимый интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML-документов. Стандартизирована W3C.
Документ в DOM представляет собой дерево узлов. Узлы связаны между собой отношением «родитель-потомок». Используется для динамического изменения страниц HTML.

BOM (Browser Object Model / объектная модель браузера) - набор объектов, описывающих содержимое документа. BOM уникальна для каждого браузера, что вносит проблемы при создании межбраузерных приложений. Поэтому Веб-консорциум предложил объектную модель документа (DOM), являющуюся стандартным способом представления веб-страниц с помощью набора объектов. В отличие от DOM объектная модель браузера (BOM) содержит набор объектов связанных непосредственно с браузером, объектов позволяющих управлять окнами (открывать, перемещать, менять размер, закрывать), строкой состояния браузера, cookie-наборами и т.п., которых нет в DOM. При написании приложений в целях поддержки межбраузерной переносимости необходимо придерживаться стандартов DOM, а к BOM прибегать лишь при крайней необходимости. Такая необходимость может возникнуть, например, при управлении окнами, строкой состояния, при необходимости узнать разрешение экрана пользователя (объект screen), или информацию о браузере (объект navigator) или данные о документе (объект location) и т.п.

Возможности BOM:
управление фреймами, поддержка задержки в исполнении кода и зацикливания с задержкой, системные диалоги,
управление адресом открытой страницы, управление информацией о браузере, управление информацией о параметрах монитора, ограниченное управление историей просмотра страниц, поддержка работы с HTTP cookie.

ServletContext
Интерфейс ServletContext объявляет методы, которые сервлет применяет для связи с контейнером сервлетов и позволяет получать информацию о среде выполнения, а также использовать ресурсы совместно с другими объектами приложения. Каждому сервлету ставится в соответствие единственный объект, реализующий ServletContext. Контекст выполнения сервлета дает средства для общения с сервером. В частности, можно получить информацию о MIME-типе файла, добавить/удалить атрибуты контекста или записать информацию в log-файл. Получить ссылку на объект ServletContext можно вызовом метода getServletContext (). Используя объект ServletContext, можно регистрировать события сервлета, сессии и запроса.
Следующие методы позволяют получить из контекста сервлета базовую информацию:
String getMimeType (String filename) – определение MIME-типа файла или документа. По умолчанию MIME-типом для сервлетов является text/plain, но используется обычно text/html;
String getRealPath (String filename) – определение истинного маршрута файла относительно каталога, в котором сервер хранит документы;
String getServerInfo () – предоставляет информацию о самом сервере.
Ряд методов предназначен для управления атрибутами, с помощью которых передается информация между различными компонентами приложения (JSP, сервлетами):
Object getAttribute(String name) – получает значение атрибута по имени;
Enumeration getAttributeNames() – получает список имен атрибутов;
void setAttribute(String name, Object object) – добавляет атрибут и его значение в контекст;
void removeAttribute (String name) – удаляет атрибут из контекста;
ServletContext getContext(String uripath) – позволяет получить доступ к контексту других ресурсов данного контейнера сервлетов;
String getServletContextName()– возвращает имя сервлета, которому принадлежит данный объект интерфейса ServletContext.

HTTP-запрос. Структура запроса http
Запрос - это сообщение, посылаемое клиентом серверу. Первая строка этого сообщения включает в себя метод, который должен быть применен к запрашиваемому ресурсу, идентификатор ресурса и используемую версию протокола.
Строка Статус начинается со строки с названием метода, за которым следует URI-Запроса и использующаяся версия протокола. В поле Метод указывается метод, который должен быть применен к ресурсу, идентифицируемому URI-Запроса.
Стартовая строка:
Метод URI HTTP/Версия
GET /spip.html HTTP/1.1
Заголовки:
Host: cs.ifmo.ru
User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru;
rv:1.9b5) Gecko/2008050509 Firefox/3.6
Accept: text/html
Connection: close
Тело сообщения

Трансляция JSP
На фазе трансляции каждый тип данных в странице JSP интерпретируется отдельно. Шаблонные данные трансформируются в код, который будет помещать данные в поток, возвращающий данные клиенту. Элементы JSP трактуются следующим образом:
Директивы, используемые для управления тем, как Web-контейнер переводит и выполняет страницу JSP.
Скриптовые элементы вставляются в класс сервлета страницы JSP.
Элементы в форме <jsp:XXX ... /> конвертируются в вызов метода для компонентов JavaBeans или вызовы API Java Servlet.

Фаза трансляции может порождать ошибки, которые будут выведены только, когда страница будет в первый раз запрошена. Если ошибка возникает при трансляции страницы (например, транслятор находит элемент JSP с неправильным форматом), сервер возвращает ParseException, и исходный файл класса сервлета будет пустым или незаконченным. Последняя незаконченная строка дает указатель на неправильный элемент JSP.
Когда страница оттранслирована и откомпилирована, сервлет страницы JSP в основном следует жизненному циклу сервлета.
Если экземпляр сервлета страницы JSP не существует, контейнер:загружает класс сервлета страницы JSP;создает экземпляр класса сервлета; инициализирует экземпляр сервлета вызовом метода jspInit.
Вызывает метод _jspService, передавая ему объекты запроса и отклика. Если контейнеру нужно удалить сервлет страницы JSP, он вызывает метод jspDestroy.

Преимущества и недостатки сервлетов по сравнению с CGI
Сервлеты – это серверные сценарии, написанные на Java. Жизненным циклом сервлетов управляет веб-контейнер (или контейнер сервлетов),запросы обрабатываются в отдельных потоках на веб-контейнере.
CGI — механизм вызова пользователем программ на стороне сервера. Данные отправляются программе посредством HTTP-запроса, ормируемого веб-браузером. То, какая именно программа будет вызвана,обычно определяется URL запроса. Каждый запрос обрабатывается отдельным процессом CGI-программы.

Преимущества сервлетов:
- Выполняются быстрее, чем CGI-сценарии.
- Хорошая масштабируемость.
- Надёжность и безопасность (реализованы на Java).
- Платформенно-независимы.
- Множество инструментов мониторинга и отладки.

Недостатки сервлетов:
- Слабое разделение уровня представления и бизнес-логики
- Возможны конфликты при параллельной обработке запросов

Достоинства CGI:
- Программы могут быть написаны на множестве языков программирования.
- «Падение» CGI-сценария не приводит к «падению» всего сервера.
- Исключены конфликты при параллельной обработке нескольких запросов.
- Хорошая поддержка веб-серверами.

Недостатки CGI:
- Высокие накладные расходы на создание нового процесса.
- Плохая масштабируемость.
- Слабое разделение уровня представления и бизнес-логики.
- Могут быть платформенно-зависимыми.

Объявления и скриплеты
Выражения Java вычисляются, конвертируются в строку и вставляются в страницу. Эти вычисления происходит во время выполнения (то есть при при запросе страницы), а потому существует полный доступ к информации о самом запросе. Например, следующий код служит для отображения даты и времени запроса данной страницы:
Текущее время: <%= new java.util.Date() %>

Объявления JSP позволят вам задать методы или поля, для вставки в тело класса сервлета (вне метода service, обрабатывающего запрос). Они имеют следующую форму:<%! Код на Java %>
Поскольку объявления не осуществляют вывода, обычно они используются совместно с JSP выражениями или скриплетами. В приведенном в качестве примера фрагменте JSP отображается количество запросов к данной странице с момента загрузки сервера (или с момента последнего изменения и перезагрузки сервлета):
<%! private int accessCount = 0; %>
Количество обращений к странице с момента загрузки сервера:<%= ++accessCount %>
Также как и в скриплетах, если вам необходимо использовать последовательность символов "%>", используйте для этого последовательность "%\>". XML эквивалентом <%! Код %> является
<jsp:declaration>Код</jsp:declaration>

Скриптлеты JSP используются для помещения в них любых фрагментов кода, которые допустимы для скриптового языка, используемого в странице. Код добавляется в метод service. Синтаксис скриптлета следующий:
<%scripting language statements%>
Если скриптовый язык установлен java, скриптлет преобразуется в операторы языка Java и вставляется в сервисный метод сервлета страницы JSP. Переменные языка программирования, созданные внутри скриптлета, доступны откуда угодно внутри страницы JSP.