



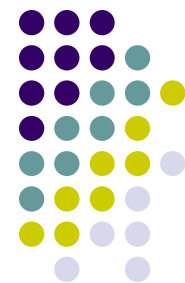
Мрежово програмиране

ASP.NET, PHP, JSP



Най-често, когато се използва сървърно ориентирана обработка, се използват многослойни Интернет страници, включващи:

- слой за база от данни (съхраняване, избор, модификация и изтриване, свързани с решаването на приложната задача от приложението)
- среден слой – сървърни програми, осъществяващи бизнес логиката (правила, алгоритми за реакция на приложението на потребителски действия или на вътрешни събития, правила за обработка на данните)
- презентационен слой за визуализиране при клиента (отразява взаимодействието с потребителя: натискане на бутони, движение на мишката, изобразяване на графични обекти, извеждане на резултати от търсене и т.н.)



ASP (Active Server Page)

- ASP е приносът на Microsoft в общността на скриптовете за сървърна обработка.
- Поддръжката на техническата документация, предоставяна от Microsoft, прави ASP добър избор за разработчици, които използват уеб сървъри на Microsoft.
- Като съставна част от уеб услугите, ASP работи бързо и има универсална приложимост.
- Недостатък на ASP е, че малко от характерните му приложения са валидни за сървърите, които не са на Microsoft.



ASP е технология на Microsoft - комбинираща HTML, скриптове и сървърни компоненти в един файл, наречен Active Server Page (ASP)

- Когато сървърът получава заявка за един ASP файл – той първо търси компилирана страница за изпълнение.
- Ако страницата не е компилирана - тогава сървърът я компилира и изпълнява.
- Резултатът от изпълнението на един ASP файл е окончателната веб страница, която се връща към браузъра на клиента.



Една ASP може да се създаде с HTML, Jscript, VBScript и други.

Посредством скриптове ASP може да има достъп до сървърните компоненти. Тези компоненти могат да се програмират на различни езици, които поддържат COM интерфейс.

Един от основните проблеми на ASP е смесването на HTML с бизнес логиката, което прави страницата трудна за разбиране, поддръжка и дебъгване. Файлът става неимоверно голям и сложен, което забавя целия процес на разработване на приложението.



Към настоящия момент технологията ASP се счита за остаряла и е заменена от ASP.NET

ASP.NET е част от средата за разработка .NET Framework. Приложения за ASP.NET могат да се пишат на всички езици за програмиране, които се компилират до Common Intermediate Language (CIL) код.

Когато се компилира .NET код в CIL код, се получава независим от операционната система код.

И вече при следващата стъпка от CIL код с JIT (Just-In-Time) компилатор .NET Framework преобразува CIL кода в машинен език, специфичен за конкретната операционна система и чак тогава .NET приложението може да бъде изпълнено на тази система.



Разделяне на визуализацията от бизнес логиката

- Смесването на HTML и изпълним код, прави уеб страниците трудни за разбиране, поддръжка и тестване
- Програмирането за клиентския интерфейс се разделя на две различни части
 - За визуализация се използва HTML код, записан във файл с разширение .aspx
 - Бизнес логиката се дефинира в отделен файл (с разширение .cs за C#), съдържащ конкретната имплементация чрез определен език
- Файлът, съдържащ бизнес логиката, се нарича изпълним код на уеб формата (Code Behind)

Архитектура на ASP.NET

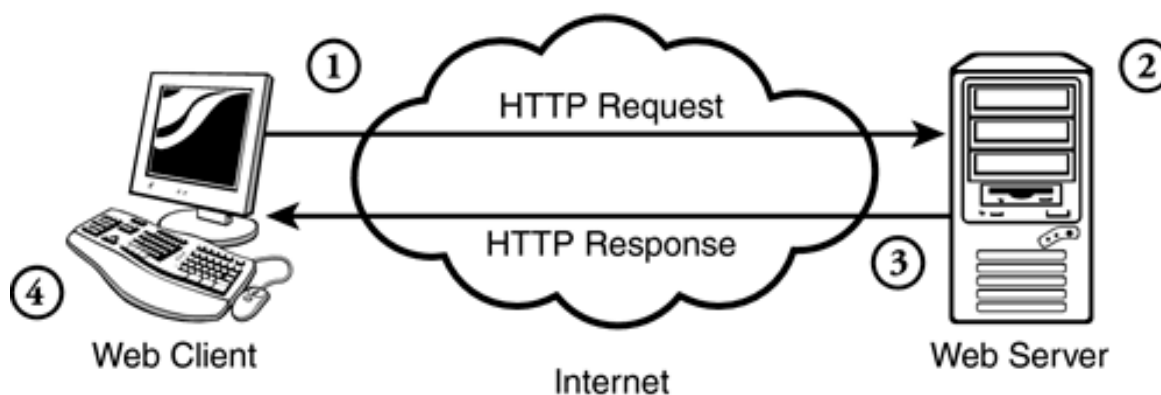


- ASP.NET е съвкупност от класове, които работят съвместно, за да обслужват HTTP заявки
- Всичко е клас, зареден от асембли (.DLL файлове)
- Работният процес на ASP.NET е отделен самостоятелен процес от IIS
- Страниците в ASP.NET са винаги компилирани до .NET класове
 - Съдържат се в асемблита



Изпълнение на ASP.NET уеб приложение

- ASP.NET уеб приложенията се изпълняват чрез серия от HTTP заявки и отговори между клиентските браузъри и уеб сървъра



Изпълнение на ASP.NET веб приложение



1. Потребителят заявява дадена страница от веб сървъра, като написва нейният адрес (URL) в брауъра. Брауърът изпраща HTTP заявка към сървъра.
2. Сървърът получава заявката, анализира я и я обработва. В случая за ASP.NET, IIS открива процеса, който може да обработи дадената заявка.
3. Резултатът от вече обработената заявка се изпраща обратно към потребителя (клиента) под формата на HTTP отговор.
4. Брауърът прочита отговора и го показва като веб страница.



Уеб форма

- Програмируема уеб страница
- Служи за потребителски интерфейс на ASP.NET приложение
- Състои се от HTML, код и контроли, които се изпълняват на уеб сървър
- Потребителят вижда резултата, като получава HTML, генериран от уеб сървър
- Кодът и контролите, които описват уеб формата, не напускат сървър

Директиви



- Предоставят възможност да се контролират много опции, влияещи върху компилацията и изпълнението на веб формата
- Важни директиви:
 - **@Page** – главна директива за формата
 - **@Import** – въвежда даден namespace във формата
 - **@Assembly** – свързва асембли с формата, когато бъде компилирана
 - **@OutputCache** – контролира способността за кеширане на формите
 - **@Register** – регистрира контрол за употреба в веб форма



ASP обекти

Събитието уеб заявка се обработва с помощта на следните обекти:

- Response – използва се за запис на данните в HTTP отговора, връщан на клиента
- Application – съдържа параметрите и конфигурациите за настройката на работата на ASP за дадения уеб сайт
- Request – съхранява съдържимото на HTTP заявката и осигурява помощни функции за обработка на данните от HTTP заявката
- Server – съдържа информация за уеб сървъра, уеб сайта, а също така осигурява поддръжка на извикващата програма
- Session – Представява състоянието на зададения уеб сеанс със зададен хост на клиента

Вградени обекти в ASP.NET



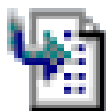
- Application (HttpApplication class)
- Session (HttpSession class)
- Request (HttpRequest class)
- Response (HttpResponse class)
- Server (HttpServerUtility class)
- Context (HttpContext class) - обект, който съдържа всички сведения за текущата заявка, известни на ASP.NET
- Cache (System.Web.Caching.Cache class)
Пространството на имената System.Web.Caching предоставя класове за кеширане на често използваните данни на сървъра.
Това пространство на имена включва класа Cache, който позволява съхраняването на такива обекти от данни, като хеш таблици и съвкупност от данни.



Компоненти на ASP.NET



Web Forms – доставят интерфейса за ASP.NET приложение



Code-behind – асоциират се с уеб форми и съдържат server-side код



Web.config – файл, съдържащ конфигурацията на ASP.NET приложението



Machine.config – файл с глобални настройки за уеб сървъра



Global.asax – файл, съдържащ код за прихващане на application-level събития



Вграждането на сървърно-базиран код в стандартна HTML страница се извършва лесно. Обикновено текстовият файл на уеб страниците, съдържащи скриптов код, който трябва да се интерпретира от уеб сървър се записват със специфично разширение, в зависимост от типа на използвания скриптов език.

Ето един пример за използване на ASP:

ASP код в HTML страница

Една обикновена HTML страница: `hello_world.html` има вида:

```
<html>
<head><title>Hello World</title></head>
<body>
    Hello World
</body></html>
```




Когато е налице **IIS (Internet Information Server)** или друг уеб сървър, поддържащ ASP, горната програма може да бъде записана с използване на сървърно-ориентиран скрипт. Създава се текстов файл, който трябва да има разширение '.asp', например hello_world.asp:

```
<%@ Language=VBScript%>
<html>
<head><title>Hello World</title></head>
<body>
    <% Response.Write("Hello World")%>
</body></html>
```



ASP.NET MVC е платформа, създадена от Microsoft, която служи за изработване на уеб приложения, използвайки модела Model-View-Controller (MVC).

Платформата използва C#, HTML, CSS, JavaScript и БД

- ASP.NET MVC е съвременно средство за изграждане на уеб приложения, което не замества изцяло уеб формите.
- Платформата включва нови тенденции в разработката на уеб приложения, притежава много добър контрол върху HTML и дава възможност за създаване на всякакви приложения.
- Моделът представлява част от приложението, което реализира бизнес логиката, също известна като домейн логика. Домейн логиката обработва данните, които се предават между базата от данни и потребителския интерфейс.
- Предимството на MVC пред класическите уеб форми е в по-детайлния контрол върху генерирания html код. Размерът на страниците е значително по-малък, но времето за разработване на приложение с MVC е по-голямо.



- **PHP (Hypertext Preprocessor).** PHP работи практически на всеки уеб сървър с малки различия в кода.
- Най-голямото предимство на PHP е, че е безплатен и с отворен изходен код. Той не изисква специален сървър както при ASP. Той е по-бърз от JSP и по-лесен за изучаване от Perl.
- В момента PHP се използва от милиони домейни по целия свят и популярността му продължава да расте. С PHP може да създават и редактират файлове, да се събира и обработва информация от формуляри, да се изпращат данни с електронна поща, да се управляват записи в бази от данни, да се съхраняват данни в променливи по време на сесия и други.



Обектно-ориентиран стил в PHP

- Обектно-ориентираният стил в PHP не се различава съществено от този на други програмни езици. Той се появява с PHP5.
- Дефиниране на клас

```
<?php
class phpClass {
    var $var1;
    var $var2 = "constant string";

    function myfunc ($arg1, $arg2) {
        [...]
    }
    [...]
}
```

- Извикване на функции от класа

```
$physics->setTitle( "Physics for High School" );
$chemistry->setTitle( "Advanced Chemistry" );
$maths->setTitle( "Algebra" );

$physics->setPrice( 10 );
$chemistry->setPrice( 15 );
$maths->setPrice( 7 );
```

- Създаване на обекти

```
$physics = new Books;
$maths = new Books;
$chemistry = new Books;
```

- Всички член данни и функции по подразбиране са public.

- Примерен клас:

```
<?php
class Books {
    /* Member variables */
    var $price;
    var $title;

    /* Member functions */
    function setPrice($par){
        $this->price = $par;
    }

    function getPrice(){
        echo $this->price . "<br/>";
    }

    function setTitle($par){
        $this->title = $par;
    }

    function getTitle(){
        echo $this->title . " <br/>";
    }
}
```



- Конструктори

```
function __construct($par1, $par2 ...){  
    $this->... = $par1  
    $this->... = $par2  
    ...  
}
```

- Деструктори

```
function __destruct($par1, $par2 ...){  
    // тук се освобождават всички заети ресурси  
}
```

- Наследяване

- Класовете-деца автоматично наследяват всички член данни и член функции от родителския клас, като функциите работят точно така, както и в родителския.

- Пример:

```
class Novel extends Books {  
    var $publisher;  
  
    function setPublisher($par){  
        $this->publisher = $par;  
    }  
  
    function getPublisher(){  
        echo $this->publisher. "<br />";  
    }  
}
```



- **Private** members – тези член данни/функции могат да се достъпват само в обхвата на класа, в който се намират. Те не могат да се извикват/променят/използват от други класове дори да наследяват въпросния клас.
- **Public** members – те могат да бъдат достъпни:
 - Извън класа, в който са декларирани
 - Вътре в класа, в който са декларирани
 - Всеки друг клас, който използва класа, в който са декларирани.
- **Protected** members – член данните/функциите от такъв вид са достъпни само за класа, в който са декларирани и класове, които наследяват този клас.
- **Function Overriding** – ако дефинираме функция в клас-дете, която има същото име, като функция, наследена от родителски клас, новата имплементация на функцията в клас-дете ще измести старата от наследения клас.



- Абстрактни класове – това са класове, от които не могат да се създават обекти. Те се използват единствено за наследяване. Когато се наследява абстрактен клас всеки метод, който е отбелязан като абстрактен, трябва да се декларира в класа-дете със същата видимост.

- Пример:

```
abstract class MyAbstractClass {  
    abstract function myAbstractFunction() {  
    }  
}
```

- Статични член данни и функции – като се декларират статични член данни или функции става възможно те да се използват без да се създава инстанция от този клас.

```
<?php  
class Foo {  
    public static $my_static = 'foo';  
  
    public function staticValue() {  
        return self::$my_static;  
    }  
}  
  
print Foo::$my_static . "\n";  
$foo = new Foo();  
  
print $foo->staticValue() . "\n";  
?>
```



PEAR – PHP Extension and Application Repository

(хранилище за разширение и приложения)

- PEAR пакетът е архив, който се състои от source код, написан на PHP.
- PEAR предоставя:
 - Структурирана библиотека с open-source код
 - Система за дистрибуция на код и пакети
 - Стандартен стил на код, написан на PHP
 - PECL – PHP Extension Community Library



- Кодът в PEAR е разпределен в „пакети“. Всеки от тези пакети се обуславя като отделен проект, който си има собствени екип от разработчици, номер на версия, документация и отношения с други пакети.
- Отношенията между пакетите не е по подразбиране, то трябва да бъде зададено. Например: „HTTP_Post“ е независим спрямо „HTTP“, т.е. не се определя от подобни имена.
- PEAR package manager дава възможност да си инсталират, деинсталират или обновяват PEAR пакетите. Преди инсталиране на пакет може да се задават зависимостите на този пакет с други пакети и тогава се инсталират по подразбиране нужните пакети.
- PEAR package manager се стартира от командният прозорец с командата „pear“.























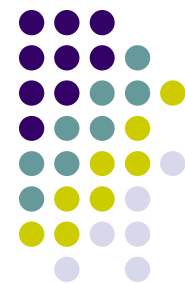
PHP accelerators

- PHP accelerator е разширение на PHP, създаден с цел да се подобри бързодействието на приложенията написани на PHP.
- Начин на работа - PHP ускорителите кешират компилирания bytecode от написаният PHP код. След това кешираният код се съхранява в общата памет и се изпълнява директно от там. Последното съществено намалява четенето на информация от диска и копирането на информация. Всичко това обаче е за сметка на паметта.
 - Практична полза от PHP accelerators – ускоряват бързодействието приложенията от 2 до **7** пъти.

Най-популярните езици за програмиране (tiobe-index)



Dec 2021	Dec 2020	Change	Programming Language		Ratings	Change
1	3	▲		Python	12.90%	+0.69%
2	1	▼		C	11.80%	-4.69%
3	2	▼		Java	10.12%	-2.41%
4	4			C++	7.73%	+0.82%
5	5			C#	6.40%	+2.21%
6	6			Visual Basic	5.40%	+1.48%
7	7			JavaScript	2.30%	-0.06%
8	12	▲		Assembly language	2.25%	+0.91%
9	10	▲		SQL	1.79%	+0.26%
10	13	▲		Swift	1.76%	+0.54%
11	9	▼		R	1.58%	-0.01%
12	8	▼		PHP	1.50%	-0.62%
13	23	▲		Classic Visual Basic	1.27%	+0.56%
14	11	▼		Groovy	1.23%	-0.30%
15	15			Ruby	1.16%	-0.01%
16	18	▲		Delphi/Object Pascal	1.14%	+0.27%
17	32	▲		Fortran	1.04%	+0.59%
18	14	▼		Perl	0.96%	-0.24%
19	16	▼		Go	0.95%	-0.19%
20	17	▼		MATLAB	0.92%	-0.18%



Какво е JSP

- JSP – Java Server Pages
- Използва се като:
 - Алтернатива на сървлетите
 - В комбинация със сървлетите
- Опростява аспекта на сървлет програмирането (това е писане на html към поток).
- Позволява да се смесва Java код директно с html – резултатът се записва към потока автоматично.



- JavaServer Pages (JSP) е технология:
 - За създаване на динамични уеб приложения
 - Позволяваща използването на Java код в HTML страници
 - Java кода се изпълнява на сървъра по време на интерпретирането на JSP страницата
 - Чист HTML се изпраща на брауъра на клиента в резултат от интерпретирането на JSP страницата



JSP технология

- JSP технологията осигурява лесен начин за разработване на динамични уеб приложения
 - Подобно на сервлетите работи в режим request/response
 - Дава възможност за генериране на динамично съдържание без да се изискват задълбочени познания за Java
 - За запознатите с езика се осигурява възможността за лесното му използване заедно със стандартния HTML
 - Позволява използването на различни XML тагове



JSP Scripting Elements

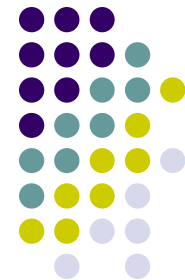
- JSP файловете се преобразувани в сървлети. Това помага да се разбере как работят скрипт JSP елементите
- *Изразите се изчисляват и автоматично се поставят в изхода от сървлета (няма нужда от out.println).*



Писане на JSP

JavaServer Pages се състоят от HTML страници със специални тагове, които позволяват включването на Java логика

- JSP скриптинг елементи - директно в html документа



Скрипт елементи

- *Изрази:* `<%= expression %>`
- *Скриплету:* `<% code %>`
- *Декларации:* `<%! code %>`
- *Директиви:* `<%@ ... %>`
- Изрази - като низ за включване към изхода на страницата.
- Скриплету - Java код, вграден в страницата.
- Декларации - общи декларации на променливи и методи.
- Директиви - информация за страниците.



Стандартни обекти в JSP

Съгласно стандарта за Java Server Pages във всички JSP страници автоматично се създават следните обекти:

- **request** – за достъп до HTTP заявката и параметрите, които клиентът е изпратил към нея
- **response** – за управление на отговора на HTTP заявката
- **out** – изходен текстов поток за отговора на HTTP заявката
- **session** – за управление на потребителските сесии
- **application** – за достъп до данните, съхранявани в контекста на уеб приложението



- **request**

- `HttpServletRequest` се свързва с променливата заявката на клиента
- Осигурява достъп до параметрите на request-a, HTTP headers, cookies и др.

- **response**

- `HttpServletResponse` се свързва с отговора, който се връща на клиента
- Могат да се задават HTTP status кодове и response header-и



- **out**
 - **PrintWriter** , който се използва за да се изпрати отговора на клиента **session**
- **session**
 - **HttpSession** обекта се сързва със заявката на клиента, т.е. с **request** обекта
 - Сесиите се създават автоматично
 - Пази статична информация за текущия потребител, свързан с тази сесия



- **application**

- `ServletContext`, който се достъпва чрез `getServletConfig().getContext()`
- Пази информация, достъпна за цялото приложение.
- Всички сървлети и JSP страници могат да си обменят информация чрез този обект

- **pageContext**

- Обвива всички останали предефинирани обекти в JSP страниците (**request**, **response**, **session**, ...) и ги пази в инстанция на **PageContext**



- **page**
 - Синоним на обекта `this` object
- **exception**
 - Дефинира обекта от тип **Throwable**
 - Достъпен е само за дефинираните error pages
 - Съдържа последното хвърлено изключение
- **config**
 - Съдържа **ServletConfig** за текущата JSP страница
 - Използва се за достъп до `init` параметрите

Обект `application`



- Използването на `application` обекта трябва да става в `synchronized` част от кода

```
synchronized (application) {  
    Vector items = (Vector)  
        application.getAttribute("items");  
    if (sharedItems == null) {  
        sharedItems = new Vector ();  
        application.setAttribute("items", items);  
    }  
}
```

- Обектът е общ за всички нишки
- Уеб контейнерите стартират самостоятелна нишка за всеки клиент



Пренасочване на клиента към друго URL

- Client redirection

```
response.sendRedirect(<url>) ;
```

- Пренасочва уеб браузъра на клиента към предварително посочено URL
- В действителност изпраща HTTP response код 302 (*Resource moved temporarily*)
- Браузърът изпраща заявка към новия адрес
- Пример:

```
response.sendRedirect("date.jsp") ;
```


Пренасочване на сървъра към друг ресурс



- Server redirection

```
request.getRequestDispatcher(<url>).  
forward(request, response)
```

- Връща съдържанието на някакъв ресурс на сървъра
- За клиента това пренасочване става неявно

- Пример:

```
request.getRequestDispatcher("date.jsp").  
forward(request, response);
```

Пример - Date JSP



- Примерната страница показва текущата дата и час

date.jsp

```
<html>
  <head><title>Date JSP
example</title></head>
  <body>
    The date is:
    <% out.println(new
java.util.Date()); %>
  </body>
</html>
```

JSP изрази



- JSP изразите се използват за директно показване на резултат от израз на Java
- Имат следният вид:

```
<%= Java израз %>
```

- Примери:

```
The time is: <%= new java.util.Date() %>
```

```
The square root of 2 is: <%= Math.sqrt(2) %>
```

```
The value of PI is: <%= Math.PI %>
```



Предефинирани променливи

- JSP имат няколко предефинирани променливи
 - `request` – еквивалент на `HttpServletRequest`
 - `response` – `HttpServletResponse`
 - `session` – `HttpSession` (свързва се с `request`)
 - `out` – използва за показване на резултата генериран от JSP (`PrintWriter`)
- Тези променливи са инициализирани и могат да се използват директно на кое да е място в JSP страницата

Пример – JSP скриплети

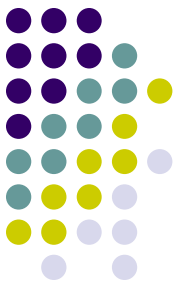


- Пример на директно използване на Java код в JSP страница:

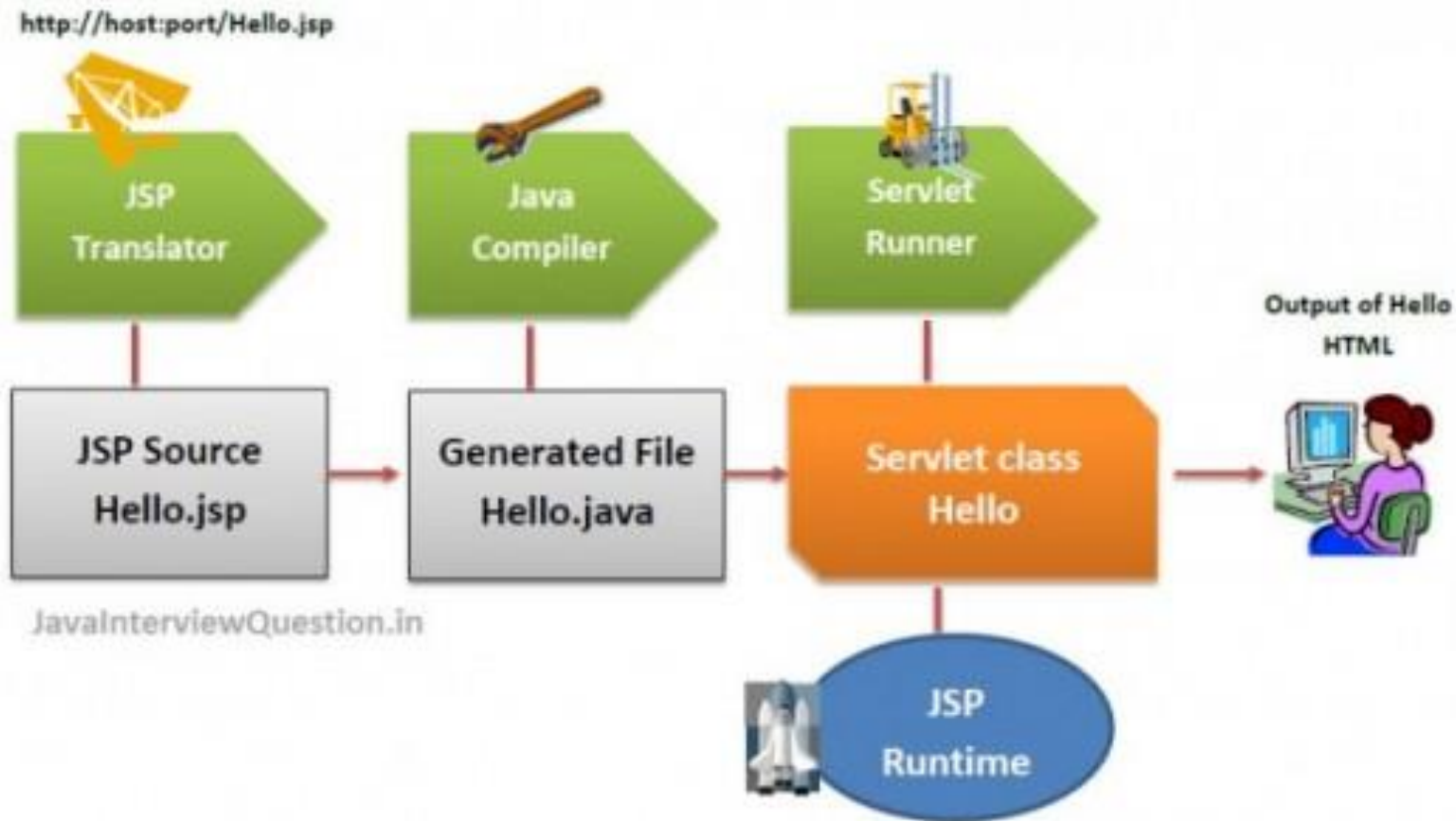
```
<% if (Math.random() < 0.5) { %>
    Have a <B>nice</B> day!
<% } else { %>
    Have an <B>interesting</B> day!
<% } %>
```

- Пример, илюстриращ използването на цикъл:

```
<% for (int i=0; i<10; i++) { %>
    <%= i %> * <%= i %> = <%= i*i %>
    <br>
<% } %>
```



Уеб контейнерът превръща JSP страниците в Java сървлети (с разширение .java), след което ги компилира до .class файлове



JSP декларации



- JSP декларациите позволяват дефинирането на методи и полета, които стават част от основното тяло на съответния сървлет.

Имат следния вид, например:

```
<%! Java code (fields and methods) %>
```

```
<%!  
    long counter = 0;  
    public void getCounter() { return counter; }  
%>
```



- Декларациите не генерират резултат
 - Обикновено се използва заедно с JSP изрази и скриплети.
- Пример:
 - Извежда на екрана колко пъти е била достъпвана страницата от момента на качването ѝ на сървъра:

```
<%! private static int accessCount = 0; %>  
This page has been accessed  
<%= ++accessCount %> times.
```


JSP директиви



JSP directive засяга цялостната структура на сървлета, до който е компилирана JSP страницата

- Има следната форма:

```
<%@ directive attribute="value" %>
```

- Може да има повече от един атрибути:

```
<%@ directive attribute1="value1"  
               attribute2="value2"  
               ...  
               attributeN="valueN" %>
```

Директивата *@page*



- *page* директивата позволява дефинирането на един или повече, видими в рамките на една страница, атрибути:
 - Указва кои пакети се `import`-ват

```
import="package.class" or  
import="package.class1, ..., package.classN"
```

- Пример:

```
<%@ page import="java.util.*" %>
```

- Атрибута `import` е единственият, който може да се повтаря многократно



- Указва използвания MIME тип за показване на извеждането (по подразиране това е "text/html")

```
contentType="MIME-Type" or  
contentType="MIME-Type; charset=Character-Set"
```

- Например директивата:

```
<%@ page contentType="text/plain" %>
```

има същия резултат като скриплета:

```
<% response.setContentType("text/plain"); %>
```



- Дефинира дали страница ще работи със *session* обекта (по подразбиране е true)

```
session="true|false"
```

- Дефинира URL на страницата, към която ще се пренасочват всички неприхванати изключения

```
errorPage="url"
```

- Декларира текущата страница като error page (позволява достъп до обекта **exception**)

```
isErrorPage="true|false"
```

Директивата *@include*

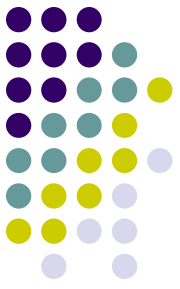


- Позволява включването на файлове по време на превръщането на страницата в сървлет (static include)
- Директивата има следния вид:

```
<%@ include file="relative url" %>
```

- Зададеното URL се разглежда като относителен път до JSP, към която сочи
- Пример:

```
<%@ include file="/include/menu.jsp" %>
```



Използване на JSP *@include*

- Например може да използваме JSP `@include`, за да добавим navigation bar към всяка страница

```
<html>
  <body>
    <%@ include file="/navbar.html" %>

    <!-- Part specific to this page ... -->

  </body>
</html>
```



Динамичен Include

- Динамично включване на страница (dynamic include):

```
<jsp:include page="header.jsp" />
```

- Резултът от включената страница се добавя към главната като се изчислява всеки път

```
<% String headerPage = "header.jsp"; %>  
<jsp:include page="<%= headerPage %>" />
```

Page Elements

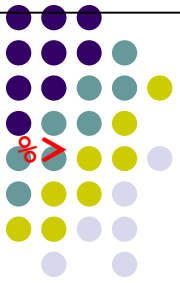
- Standard Tags
- 'Use Bean'
- Session Management
- Alternate Languages
- Custom Tags

```
...<BODY>
<%@ page language="java"
        import="com.acme.app.*, java.util.*"
        %>

<jsp:useBean id="hist"
              scope="session"
              class="com.acme.app.histBean" />

<%
    Iterator itOrders =
hist.getOrderList().iterator();
    while (itOrders.hasNext()){
        Order odr = (Order)itOrders.next();
    %>

        <TR>
        <TD>Autrags-Nr.: </TD>
        <TD> <%= odr.getRenderer("odrNum").disp() %>
    </TD>
        <TD>Liefer-Datum:</TD>
        <TD> <%= odr.getRenderer("dvDt").disp() %>
    </TD>
        </TR>
        <%
    }
    %>
</BODY></HTML>
```





Предимства

- Добра производителност
- Независимост от платформата
- Лесен развой
- Разширяемост с JavaBeans и потребителски тагове