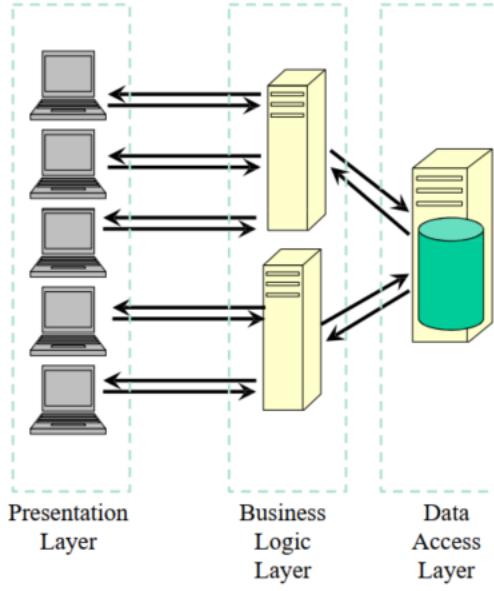


3-Tier Architecture

- Database Tier (Data Access Layer)
 - Stores and accesses data in low-level
- Server Tier (Business Logic Layer)
 - Manages application connections and process data
- Client Tier (Presentation Layer)
 - Provides interface and processing



3-Tier Architecture Advantages

- Centralized Database can be accessed by many servers at the same time
- Allow load balance of user connections on many application servers
- **Data Access Layer** is consistently designed with hardware in order to serve specific its tasks:
 - Data manipulations: update, insert, remove, etc.
 - Need more reliable hard drives
- **Business Logic Layer** are designed to provide connection points for user connections and run multi-applications
 - Need more computing power of CPU

JSP Example

```
<html>
<head> <title> Hello JSP </title> </head>
<body>
<p> Hello World:
<%= new java.util.Date() %>
</p>
</body>
</html>
```

See also the Servlet this page is translated to

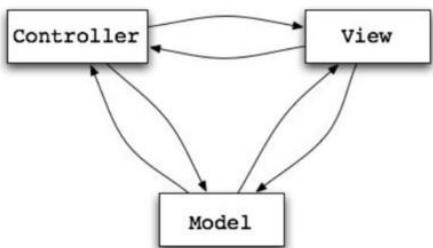
Date_jsp.java (extract)

This extract shows the part that produces the output – compare it with the JSP:

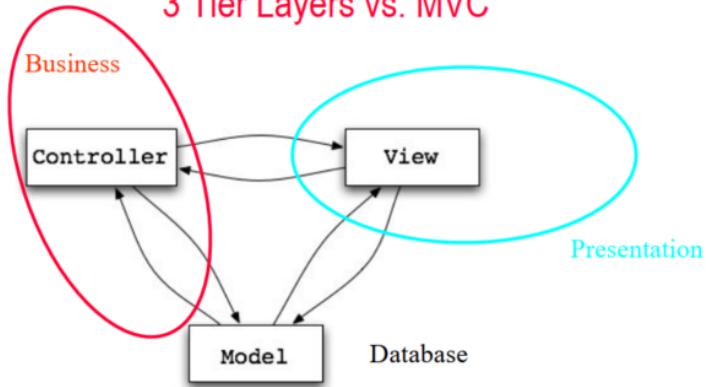
```
out = pageContext.getOut();
_jspx_out = out;
out.write("<html>\r\n");
out.write("<head> ");
out.write("<title> Hello JSP ");
out.write("</title> ");
out.write("</head>\r\n");
out.write("<body> \r\n");
out.write("<p> Hello World:\r\n      ");
out.print( new java.util.Date() );
out.write("\r\n");
out.write("</p>\r\n");
out.write("</body>\r\n");
out.write("</html>\r\n");
```

Example Control Flow in MVC

- User interacts with the **VIEW UI**
- CONTROLLER** handles the user input (often a callback function attached to **UI** elements)
- CONTROLLER** updates the **MODEL**
- VIEW** uses **MODEL** to generate new **UI**
- UI** waits for user interaction



3 Tier Layers vs. MVC



Presentation:

- View is the user interface (e.g. button)
- Controller is the code (e.g. callback for button)

Data:

- Model is the database

Basic PHP syntax

A PHP scripting block always starts with `<?php` and ends with `?>`. A PHP scripting block can be placed (almost) anywhere in an HTML document.

```

<html>
<!-- hello.php CS443 -->
<head><title>Hello World</title></head>
<body>
<p>This is going to be ignored by the PHP interpreter.</p>

<?php echo '<p>While this is going to be parsed.</p>'; ?>

<p>This will also be ignored by the PHP preprocessor.</p>

<?php print('<p>Hello and welcome to <i>my</i> page!</p>'); ?>

<?php
// This is a comment
/*
  This is
  a comment
  block
*/
?>

</body>
</html>

```

[view the output page](#)

`print` and `echo` for output

a semicolon (`;`) at the end of each statement

`//` for a single-line comment
`/* */` for a large comment block.

Scalars

All variables in PHP start with a `$` sign symbol. A variable's type is determined by the context in which that variable is used (i.e. there is no strong-typing in PHP).

```

<html><head></head>
<!-- scalars.php CS443 -->
<body> <p>
<?php
$foo = true; if ($foo) echo "It is TRUE! <br /> \n";
$txt='1234'; echo "$txt <br /> \n";
$a = 1234; echo "$a <br /> \n";
$a = -123;
echo "$a <br /> \n";
$a = 1.234;
echo "$a <br /> \n";
$a = 1.2e3;
echo "$a <br /> \n";
$a = 7E-10;
echo "$a <br /> \n";
echo "Arnold once said: \"I'll be back\", <br /> \n";
$bbeer = 'Heineken';
echo "$beer's taste is great <br /> \n";
$cstr = <<<EOB
Example of string
spanning multiple lines
using "heredoc" syntax.
EOB;
echo $cstr;
?>
</p>
</body>
</html>

```

[view the output page](#)

Four scalar types:

boolean

true or false

integer,

float,

floating point numbers

string

single quoted

double quoted

Arrays

An array in PHP is actually an ordered map. A map is a type that maps values to keys.

```
<?php
$arr = array("foo" => "bar", 12 => true);
echo $arr["foo"]; // bar
echo $arr[12]; // 1
?>
```

`array()` = creates arrays

`key` = either an integer or a string.

`value` = any PHP type.

```
<?php
array(5 => 43, 32, 56, "b" => 12);
array(5 => 43, 6 => 32, 7 => 56, "b" => 12);
?>
```

if no key given (as in example), the PHP interpreter uses (maximum of the integer indices + 1).

if an existing key, its value will be overwritten.

can set values in an array

```
<?php
$arr = array(5 => 1, 12 => 2);
foreach ($arr as $key => $value) { echo $key, '=>', $value; }
$arr[] = 56; // the same as $arr[13] = 56;
$arr["x"] = 42; // adds a new element
unset($arr[5]); // removes the element
unset($arr); // deletes the whole array
$Sa = array(1 => 'one', 2 => 'two', 3 => 'three');
unset($Sa[2]);
$Sb = array_values($Sa);
?>
```

[view the output page](#)

`unset()` removes a key/value pair

`array_values()` makes reindexing effect (indexing numerically)

[*Find more on arrays](#)

Constants

A constant is an identifier (name) for a simple value. A constant is case-sensitive by default. By convention, constant identifiers are always uppercase.

```
<?php

// Valid constant names
define("FOO", "something");
define("FOO2", "something else");
define("FOO_BAR", "something more");

// Invalid constant names (they shouldn't start
// with a number!)
define("2FOO", "something");

// This is valid, but should be avoided:
// PHP may one day provide a "magical" constant
// that will break your script

define("__FOO__", "something");
?>
```

You can access constants anywhere in your script without regard to scope.

Operators

• **Arithmetic Operators:** `+, -, *, /, %, ++, --`

• **Assignment Operators:** `=, +=, -=, *=, /=, %=`

Example	Is the same as
<code>x+=y</code>	<code>x=x+y</code>
<code>x-=y</code>	<code>x=x-y</code>
<code>x*=y</code>	<code>x=x*y</code>
<code>x/=y</code>	<code>x=x/y</code>
<code>x%y</code>	<code>x=x%y</code>

• **Comparison Operators:** `==, !=, >, <, >=, <=`

• **Logical Operators:** `&&, ||, !`

• **String Operators:** `. and .=` (for string concatenation)

```
$a = "Hello ";
$b = $a . "World!"; // now $b contains "Hello World!"

$a = "Hello ";
$a .= "World!";
```

Conditionals: if else

Can execute a set of code depending on a condition

```
<html><head></head>
<!-- if-cond.php CS443 -->
<body>

<?php
$d=date("D");
echo $d, "<br/>";
if ($d=="Fri")
    echo "Have a nice weekend! <br/>";
else
    echo "Have a nice day! <br/>";

$x=10;
if ($x==10)
{
    echo "Hello<br />";
    echo "Good morning<br />";
}

?>
</body>
</html>
```

`if (condition)`

code to be executed if condition is `true`;

`else`

code to be executed if condition is `false`;

`date()` is a built-in PHP function that can be called with many different parameters to return the date (and/or local time) in various formats

In this case we get a three letter string for the day of the week.

[view the output page](#)

Conditionals: switch

Can select one of many sets of lines to execute

```
<html><head></head>
<body>
<!-- switch-cond.php CS443 -->
<?php
$x = rand(1,5); // random integer
echo "x = $x <br/><br/>";
switch ($x)
{
case 1:
    echo "Number 1";
    break;
case 2:
    echo "Number 2";
    break;
case 3:
    echo "Number 3";
    break;
default:
    echo "No number between 1 and 3";
    break;
}
?>

</body>
</html>
```

```
switch (expression)
{
    case label1:
        code to be executed if
        expression = label1;
        break;
    case label2:
        code to be executed if
        expression = label2;
        break;
    default:
        code to be executed
        if expression is different
        from both label1 and label2;
        break;
}
```

[view the output page](#)

Looping: while and do-while

Can loop depending on a condition

```
<html><head></head>
<body>
<?php
$si=1;
while($si <= 5)
{
    echo "The number is $i <br />";
    $i++;
}
?>

</body>
</html>
```

[view the output page](#)

loops through a block of code if, and as long as, a specified condition is true

```
<html><head></head>
<body>
<?php
$si=0;
do
{
    $i++;
    echo "The number is $i <br />";
} while($i <= 10);
?>

</body>
</html>
```

[view the output page](#)

loops through a block of code once, and then repeats the loop as long as a special condition is true (so will always execute at least once)

Looping: for and foreach

Can loop depending on a "counter"

```
<?php
for ($i=1; $i<=5; $i++)
{
    echo "Hello World!<br />";
}
?>
```

loops through a block of code a specified number of times

```
<?php
$a_array = array(1, 2, 3, 4);
foreach ($a_array as $value)
{
    $value = $value * 2;
    echo "$value <br /> \n";
}
?>
```

[view the output page](#)

```
<?php
$a_array=array("a","b","c");
foreach ($a_array as $key => $value)
{
    echo $key . " = " . $value . "\n";
}
?>
```

loops through a block of code for each element in an array

User Defined Functions

Can define a function using syntax such as the following:

```
<?php
function foo($arg_1, $arg_2, /* ... , */ $arg_n)
{
    echo "Example function.\n";
    return $retval;
}
?>
```

Can also define conditional functions, functions within functions, and recursive functions.

Can return a value of any type

```
<?php
function square($num)
{
    return $num * $num;
}
echo square(4);
?>
```

```
<?php
function small_numbers()
{
    return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
echo $zero, $one, $two;
?>
```

```
<?php
function takes_array($input)
{
    echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
takes_array(array(1,2));
?>
```

[view the output page](#)

Variable Scope

The scope of a variable is the context within which it is defined.

```
<?php
$a = 1; /* limited variable scope */
function Test()
{
    echo $a;
    /* reference to local scope variable */
}
Test();
?>
```

The scope is local within functions, and hence the value of \$a is undefined in the "echo" statement.

```
<?php
$a = 1;
$b = 2;
function Sum()
{
    global $a, $b;
    $b = $a + $b;
}
Sum();
echo $b;
?>
```

global
refers to its
global
version.

```
<?php
function Test()
{
    static $a = 0;
    echo $a;
    $a++;
}
Test();
Test();
Test();
?>
```

static
does not lose
its value.

[view the output page](#)

Including Files

The `include()` statement includes and evaluates the specified file.

```
// vars.php
<?php
$color = 'green';
$fruit = 'apple';

?>

// test.php
<?php

echo "$color $fruit"; // A
include 'vars.php';
echo "$color $fruit"; // A green apple
?>
```

[view the output page](#)

```
<?php
function foo()
{
    global $color;

    include ('vars.php');

    echo "A $color $fruit";
}

/* vars.php is in the scope of foo() so      *
 * $fruit is NOT available outside of this   *
 * scope. $color is because we declared it   *
 * as global.                                */
foo();           // A green apple
echo "A $color $fruit"; // A green
?>
```

[view the output page](#)

*The scope of variables in "included" files depends on where the "include" file is added!

You can use the `include_once`, `require`, and `require_once` statements in similar ways.

PHP Information

The `phpinfo()` function is used to output PHP information about the version installed on the server, parameters selected when installed, etc.

```
<html><head></head>
<!- info.php CS443
<body>
<?php
// Show all PHP information
phpinfo();
?>
<?php
// Show only the general information
phpinfo(INFO_GENERAL);
?>
</body>
</html>
```

[view the output page](#)

INFO_GENERAL	The configuration line, php.ini location, build date, Web Server, System and more
INFO_CREDITS	PHP 4 credits
INFO_CONFIGURATION	Local and master values for php directives
INFO_MODULES	Loaded modules
INFO_ENVIRONMENT	Environment variable information
INFO_VARIABLES	All predefined variables from EGPCS
INFO_LICENSE	PHP license information
INFO_ALL	Shows all of the above (default)

Server Variables

The `$_SERVER` array variable is a reserved variable that contains all server information.

```
<html><head></head>
<body>

<?php
echo "Referer: " . $_SERVER["HTTP_REFERER"] . "<br />";
echo "Browser: " . $_SERVER["HTTP_USER_AGENT"] . "<br />";
echo "User's IP address: " . $_SERVER["REMOTE_ADDR"];
?>

<?php
echo "<br/><br/><br/>";
echo "<h2>All information</h2>";
foreach ($_SERVER as $key => $value)
{
    echo $key . " = " . $value . "<br/>";
}
?>

</body>
</html>
```

[\\$_SERVER info on php.net](#)

[view the output page](#)

The `$_SERVER` is a super global variable, i.e. it's available in all scopes of a PHP script.

File Open

The `fopen("file_name", "mode")` function is used to open files in PHP.

r	Read only.	r+	Read/Write.
w	Write only.	w+	Read/Write.
a	Append.	a+	Read/Append.
x	Create and open for write only.	x+	Create and open for read/write.

```
<?php  
$fh=fopen("welcome.txt","r");  
?>
```

For `w`, and `a`, if no file exists, it tries to create it (use with caution, i.e. check that this is the case, otherwise you'll overwrite an existing file).

```
<?php  
if  
( !($fh=fopen("welcome.txt","r")) )  
exit("Unable to open file!");  
?>
```

For `x` if a file exists, this function fails (and returns 0).

If the `fopen()` function is unable to open the specified file, it returns 0 (false).

File Workings

`fclose()` closes a file.

`feof()` determines if the end is true.

`fgetc()` reads a single character

`fgets()` reads a line of data

`fwrite()`, `fputs()`
writes a string with and without \n

`file()` reads entire file into an array

```
<?php  
$myFile = "welcome.txt";  
if (!($fh=fopen($myFile,'r')))  
exit("Unable to open file.");  
while (!feof($fh))  
{  
$x=fgetc($fh);  
echo $x;  
}  
fclose($fh);  
?>
```

[view the output page](#)

```
<?php  
$myFile = "welcome.txt";  
$fh = fopen($myFile, 'r');  
$theData = fgets($fh);  
fclose($fh);  
echo $theData; view the output page  
?>
```

[view the output page](#)

```
<?php  
$lines = file('welcome.txt');  
foreach ($lines as $l_num => $line)  
{  
echo "Line #{$l_num}:" .  
$line."<br/>"  
}  
?>
```

[view the output page](#)

```
<?php  
$myFile = "testFile.txt";  
$fh = fopen($myFile, 'a') or  
die("can't open file");  
$stringData = "New Stuff 1\n";  
fwrite($fh, $stringData);  
$stringData = "New Stuff 2\n";  
fwrite($fh, $stringData);  
fclose($fh);  
?>
```

[view the output page](#)

Form Handling

Any form element is automatically available via one of the built-in PHP variables (provided that HTML element has a "name" defined with it).

```
<html>  
<!-- form.html CS443 -->  
<body>  
<form action="welcome.php" method="post">  
Enter your name: <input type="text" name="name" /> <br/>  
Enter your age: <input type="text" name="age" /> <br/>  
<input type="submit" /> <input type="reset" />  
</form>  
</body>  
</html>
```

```
<html>  
<!-- welcome.php COMP 519 -->  
<body>  
Welcome <?php echo $_POST["name"] . ";" ?><br />  
You are <?php echo $_POST["age"]; ?> years old!  
</body>  
</html>
```

[view the output page](#)

`$_POST`
contains all POST data.

`$_GET`
contains all GET data.

Cookie Workings

`setcookie(name, value, expire, path, domain)`

creates cookies.

```
<?php  
setcookie("uname", $_POST["name"], time()+36000);  
?>  
<html>  
<body>  
<p>  
Dear <?php echo $_POST["name"] ?>, a cookie was set on this  
page! The cookie will be active when the client has sent the  
cookie back to the server.  
</p>  
</body>  
</html>
```

[view the output page](#)

NOTE:
`setcookie()` must appear
BEFORE `<html>` (or
any output) as it's part
of the header
information sent with
the page.

```
<html>  
<body>  
<?php  
if ( isset($_COOKIE["uname"]) )  
echo "Welcome " . $_COOKIE["uname"] . "!"<br />;  
else  
echo "You are not logged in!"<br />;  
?  
</body>  
</html>
```

[view the output page](#)

`$_COOKIE`
contains all COOKIE data.

`isset()`
finds out if a cookie is set

use the cookie name as a
variable

Getting Time and Date

`date()` and `time()` formats a time or a date.

```
<?php
//Prints something like: Monday
echo date("l");
//Like: Monday 15th of January 2003 05:51:38 AM
echo date("l jS \of F Y h:i:s A");
//Like: Monday the 15th
echo date("l \\t\\h\\e jS");
?>
```

[view the output page](#)

`date()` returns a string formatted according to the specified format.

```
<?php
$nextWeek = time() + (7 * 24 * 60 * 60);
// 7 days; 24 hours; 60 mins; 60secs
echo 'Now: ' . date('Y-m-d') . "\n";
echo 'Next Week: ' . date('Y-m-d', $nextWeek) . "\n";
?>
```

[view the output page](#)

`time()` returns current Unix timestamp

*Here is more on date/time formats: <http://uk.php.net/manual/en/function.date.php>

Required Fields in User-Entered Data

A multipurpose script which asks users for some basic contact information and then checks to see that the required fields have been entered.

```
<html>
<!-- form_checker.php CS443 -->
<head>
<title>PHP Form example</title>
</head>
<body>
<?php
/*declare some functions*/
```

Print Function

```
function print_form($f_name, $l_name, $email, $os)
{
?>

<form action="form_checker.php" method="post">
First Name: <input type="text" name="f_name" value="<?php echo $f_name?>" /> <br/>
Last Name <b></b><input type="text" name="l_name" value="<?php echo $l_name?>" /> <br/>
Email Address <b></b><input type="text" name="email" value="<?php echo $email?>" /> <br/>
Operating System: <input type="text" name="os" value="<?php echo $os?>" /> <br/><br/>
<input type="submit" name="submit" value="Submit" /> <input type="reset" />
</form>

<?php
} /** end of "print_form" function
```

Check and Confirm Functions

```
function check_form($f_name, $l_name, $email, $os)
{
    if (! $l_name || ! $email) {
        echo "<h3>You are missing some required fields!</h3>";
        print_form($f_name, $l_name, $email, $os);
    }
    else{
        confirm_form($f_name, $l_name, $email, $os);
    }
} /** end of "check_form" function
```

```
function confirm_form($f_name, $l_name, $email, $os)
{
?>

<h2>Thanks! Below is the information you have sent to us.</h2>
<h3>Contact Info</h3>

<?php
echo "Name: $f_name $l_name <br/>";
echo "Email: $email <br/>";
echo "OS: $os";
} /** end of "confirm_form" function
```

Main Program

```
/*Main Program*/
if (! $_POST["submit"])
{
?>

<h3>Please enter your information</h3>
<p>Fields with a "*" are required.</p>

<?php
    print_form("", "", "", "");
}
else{
    check_form($_POST["f_name"], $_POST["l_name"], $_POST["email"], $_POST["os"]);
}
?>

</body>
</html>
```

[view the output page](#)

Object oriented programming in PHP

- PHP, like most modern programming languages (C++, Java, Perl, JavaScript, etc.), supports the creation of objects.
- Creating an object requires you to first define an object class (containing variables and/or function definitions) and then using the "new" keyword to create an instance of the object class. (Note that the object must be defined before you instantiate it.)

```
<?php
// Assume that the "Person" object has been previously defined. . .

$x = new Person; // creates an instance of the Person class (*no* quotes)

// The object type need not be "hardcoded" into the declaration.

$object_type = 'Person';
$y = new $object_type; // equivalent to $y = new Person;

$z = new Vehicle('Jaguar','green'); // creating an object and passing
// arguments to its constructor

?>
```

Defining (declaring) a class

- Use the "class" keyword which includes the class name (case-insensitive, but otherwise following the rules for PHP identifiers). Note: The name "stdClass" is reserved for use by the PHP interpreter.

```
<?php
class Person
{
    var $name;

    function set_name($new_name) {
        $this->$name = $new_name;
    }

    function get_name() {
        return $this -> name;
    }
}
```

- Use the "\$this" variable when accessing properties and functions of the current object. Inside a method this variable contains a reference to the object on which the method was called.

Declaring a class (cont.)

- Properties and functions can be declared as "public" (accessible outside the object's scope), "private" (accessible only by methods within the same class), or "protected" (accessible only through the class methods and the class methods of classes inheriting from the class).
- Note that unless a property is going to be explicitly declared as public, private, or protected, it need not be declared before being used (like regular PHP variables).

```
<?php
class Person
{
    protected $name;
    protected $age;

    function set_name($new_name) {
        $name = $this -> new_name;
    }

    function get_name() {
        return $this -> name;
    }
}
```

Declaring a class (cont.)

- Classes can also have their own constants defined (using the "const" keyword), can have their own static properties and functions (using the keyword "static" before "var" or "function"), and can also have constructors and destructors (see below).
- Static properties and functions are accessed (see below) using a different format than usual for objects, and static functions cannot access the objects properties (i.e. the variable \$this is not defined inside of a static function).

```
<?php

class HTMLtable {
    static function start() {
        echo "<table> \n";
    }
    static function end() {
        echo "</table> \n";
    }
}

HTMLtable::start();

?>
```

Accessing properties and methods

- Once you have an object, you access methods and properties (variables) of the object using the `->` notation.

```
<?php  
  
$me = new Person;  
  
$me -> set_name('Russ');  
$me -> print_name();  
$name = $me -> get_name();  
echo $me -> get_name();  
  
$age = 36;  
$me -> set_age($age);  
  
?>
```

Constructors and destructors

- Constructors are methods that are (generally) used to initialize the object's properties with values as the object is created. Declare a constructor function in an object by writing a function with the name `__construct()`.

```
<?php  
class Person {  
    protected $name;  
    protected $age;  
    function __construct($new_name, $new_age) {  
        $this -> name = $new_name;  
        $this -> age = $new_age;  
    }  
    // . . . other functions here . . .  
}  
  
$p = new Person('Bob Jones', 45);  
$q = new Person('Hamilton Lincoln', 67);  
?>
```

- Destructors (defined with a function name of `__destruct()`) are called when an object is destroyed, such as when the last reference to an object is removed or the end of the script is reached (the usefulness of destructors in PHP is limited, since, for example dynamic memory allocation isn't possible in the same way that it is in C/C++).

Inheritance

- Use the "extends" keyword in the class definition to define a new object that inherits from another.

```
<?php  
class Employee extends Person {  
    var $salary;  
  
    function __construct($new_name, $new_age, $new_salary) {  
        $this -> salary = $new_salary;  
        parent::__construct($new_name, $new_age); // call the constructor  
                                         // of parent object  
    }  
    function update_salary($new_salary) {  
        $this -> salary = $new_salary;  
    }  
}$emp = new Employee('Dave Underwood', 25, 25000);  
  
?>
```

Inheritance (cont.)

- The constructor of the parent isn't called unless the child explicitly references it (as in this previous case). There is no automatic chain of calls to constructors in a sequence of objects defined through inheritance.
- You could "hard-code" the call to the parent constructor using the function call `"Person::__construct($new_name, $new_age);"` but it's typically better to define it in the manner given using the `parent::method()` notation. The same manner is used to call the method of a parent that has been overridden by its child.
- You can use the "self" keyword to ensure that a method is called on the current class (if a method might be subclassed), in this style `self::method()`.
- To check if an object is of a particular class, you can use the `instanceof` operator.

```
if ($p instanceof Employee) {  
    // do something here  
}
```

More on classes

- You can also define interfaces for objects (for which any object that uses that interface must provide implementations of certain methods), and you can define abstract classes or methods (that must be overridden by its children).
- The keyword “final” can be used to denote a method that cannot be overridden by its children.

```
class Person {  
    var $name;  
  
    final function get_name() {  
        return $this -> name;  
    }  
}
```

```
<?php  
  
// Declare the interface 'iTemplate'  
interface iTemplate  
{  
    public function setVariable($name, $var);  
    public function getHtml($template);  
}  
  
// Implement the interface  
// This will work  
class Template implements iTemplate  
{  
    private $vars = array();  
  
    public function setVariable($name, $var)  
    {  
        $this->vars[$name] = $var;  
    }  
  
    public function getHtml($template)  
    {  
        foreach($this->vars as $name => $value) {  
            $template = str_replace('{ ' . $name . ' }', $value, $template);  
        }  
  
        return $template;  
    }  
}
```

```
// This will not work  
// Fatal error: Class BadTemplate contains 1 abstract methods  
// and must therefore be declared abstract (iTemplate::getHtml)  
class BadTemplate implements iTemplate  
{  
    private $vars = array();  
  
    public function setVariable($name, $var)  
    {  
        $this->vars[$name] = $var;  
    }  
}  
?>
```

More on classes (cont.)

- There are methods for “introspection” about classes, i.e. the ability of a program to examine an object's characteristics.

For example, the function `class_exists()` can be used (surprise!) to determine whether a class exists or not.

The function `get_declared_classes()` returns an array of declared classes.

```
$classes = get_declared_classes();
```

You can also get an array of method names in any of the following (equivalent) manners:

```
$methods = get_class_methods(Person);  
$methods = get_class_methods('Person');  
$class = 'Person';  
$methods = get_class_methods($class);
```

More introspection functions

- There are a wide variety of introspection functions, several more are listed below.

```
get_class_vars($object); /* gets an associative array that maps
                           property names to values (including
                           inherited properties), but it *only*
                           gets properties that have default
                           values (those initialized with
                           simple constants) */

is_object($object); // returns a boolean value

get_class($object); /* returns the class to which an object
                      belongs */

method_exists($object, $method); // returns a boolean value

get_object_vars($object); /* returns an associative array
                           mapping properties to values (for
                           those values that are set (i.e.
                           not null) */
```

PHP sessions (cont.)

- To start a session, use the function `session_start()` at the beginning of your PHP script before you store or access any data. For the session to work properly, this function needs to execute before any other header calls or other output is sent to the browser.

```
<?php
session_start();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>Session example</title>
</head>
<body>
<?php
include_once ('object.php'); // Includes definition of the Person class
$_SESSION['hello'] = 'Hello world';
echo $_SESSION['hello'] . "<br/><br/>\n";
$_SESSION['one'] = 'one';
$_SESSION['two'] = 'two';
$me = new Person("Russ", 36, 2892700);
$_SESSION['name'] = $me->get_name();
echo "Testing " . $_SESSION['one'] . ", " . $_SESSION['two'] . ", " . $me-
  >get_number() . " . . <br/>\n";
?>
</body></html>
```

Using session variables

- Once a session variable has been defined, you can access it from other pages.

```
<?php
session_start();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>Session example 2</title>
</head>
<body>
<?php
echo "Welcome to a new page ". $_SESSION['name'] ."!<br/>\n";
echo "Hope you enjoy your stay! <br/>";
?>
<p>Back to regular HTML text...
</p>
</body>
</html>
```

[view the output page](#)

More on session variables

- You need to include a call to the `session_start()` function for each page on which you want to access the session variables.
- A session will end once you quit the browser (unless you've set appropriate cookies that will persist), or you can call the `session_destroy()` function. (Note, however, even after calling the `session_destroy()` function, session variables are still available to the rest of the currently executing PHP page.)
- The function `session_unset()` removes all session variables. If you want to remove one variable, use the `unset($var)` function call.
- The default timeout for session files is 24 minutes. It's possible to change this timeout.

Deleting all session variables

```
<?php  
session_start();  
?  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
  
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">  
  
<head>  
<title>Session example 3</title>  
</head>  
<body>  
<?php  
echo "Deleting all session variables using session_unset(); <br/>\n";  
session_unset();  
echo "Now the session variables are gone. <br/>\n";  
if (isset($_SESSION['name']))  
{ echo $_SESSION['name'] . "<br/>\n"; }  
else  
{ echo "Session variable is not here."; }  
?  
</body>  
</html>
```

[view the output page](#)

Backing up/restoring a mySQL database

- You can back up an entire database with a command such as

```
mysqldump -h mysql -u martin martin > backup.sql
```

(Run from the Unix command line.)

- This gives a script containing SQL commands to reconstruct the table structure (of all tables) and all of the data in the table(s).
- To restore the database (from scratch) you can use this type of Unix command:

```
mysql -h mysql -u martin martin < backup.sql
```

(Use with caution, as this can overwrite your database.)
- Other commands are possible to backup/restore only certain tables or items in tables, etc. if that is what you desire. For example

```
mysqldump -h mysql -u martin martin books clients> backup.sql
```

stores information about the "books" and "clients" tables in the "martin" database.

Backing up/restoring a mySQL database (cont.)

Run from the Windows command line:

- Run the mysqldump.exe program using the following arguments:

```
mysqldump.exe -e -u[username] -p[password] -h[hostname] [database name] > C:\[filename].sql
```

- Run the mysql.exe program using the following arguments.

```
mysql -u[user name] -p[password] -h[hostname] [database name]< C:\[filename].sql
```

Putting Content into Your Database with PHP

We can simply use PHP functions and mySQL queries together:

- Connect to the database server and login (this is the PHP command to do so)

```
mysql_connect("host", "username", "password");
```

- Choose the database

```
mysql_select_db("database");
```

Host:	mysql
Database:	martin
Username:	martin
Password:	----

- Send SQL queries to the server to add, delete, and modify data

```
mysql_query("query");
```

(use the exact same query string as you would normally use in SQL, without the trailing semi-colon)

- Close the connection to the database server (to ensure the information is stored properly)

```
mysql_close();
```

- Note:** For this to work properly on the URL server

Student Database: data_in.php

```
<html>
<head>
<title>Putting Data in the DB</title>
</head>
<body>
<?php
/*insert students into DB*/
if(isset($_POST["submit"])) {
    $db = mysql_connect("mysql", "martin");
    mysql_select_db("martin");

    $date=date("Y-m-d"); /* Get the current date in the right SQL format */

    $sql="INSERT INTO students VALUES(NULL,'$_POST["f_name"]','$_POST["l_name"]',$_POST["student_id"],'$_POST["email"]',$_POST["group"]'); /* construct the query */
    mysql_query($sql); /* execute the query */
    mysql_close();

    echo"<h3>Thank you. The data has been entered.</h3> \n";
    echo'<p><a href="data_in.php">Back to registration</a></p>' . "\n";
    echo'<p><a href="data_out.php">View the student lists</a></p>' . "\n";
}
}
```

Student Database: data_in.php

```
else {
?>
<h3>Enter your items into the database</h3>
<form action="data_in.php" method="post">
First Name: <input type="text" name="f_name" /> <br/>
Last Name: <input type="text" name="l_name" /> <br/>
ID: <input type="text" name="student_id" /> <br/>
email: <input type="text" name="email" /> <br/>
Group: <select name="gr">
    <option value ="1">1</option>
    <option value ="2">2</option>
    <option value ="3">3</option>
</select><br/><br/>
<input type="submit" name="submit" /> <input type="reset" />
</form>

<?php
}      /* end of "else" block */
?>

</body>
</html>
```

Getting Content out of Your Database with PHP

Similarly, we can get some information from a database:

- Connect to the server and login, choose a database

```
mysql_connect ("host", "username", "password");
```

```
mysql_select_db ("database");
```

- Send an SQL query to the server to select data from the database into an array

```
$result=mysql_query ("query");
```

- Either, look into a row and a fieldname

```
$num=mysql_numrows ($result);
```

```
$variable=mysql_result ($result,$i,"fieldname");
```

- Or, fetch rows one by one

```
$row=mysql_fetch_array ($result);
```

- Close the connection to the database server

```
mysql_close();
```

Student Database: data_out.php

```
<html>
<head>
<title>Getting Data out of the DB</title>
</head>
<body>
<h1> Student Database </h1>
<p> Order the full list of students by
<a href="data_out.php?order=date">date</a>,
<a href="data_out.php?order=student_id">id</a>, or
by <a href="data_out.php?order=l_name">surname</a>.
</p>

<p>
<form action="data_out.php" method="post">
Or only see the list of students in group
<select name="gr">
    <option value ="1">1</option>
    <option value ="2">2</option>
    <option value ="3">3</option>
</select>
<br/>
<input type="submit" name="submit" />
</form>
</p>
```

Student Database: data_out.php

```
<?php
/*get students from the DB */
$db = mysql_connect("mysql","martin");
mysql_select_db("martin", $db);

switch($_GET["order"]){
case 'date': $sql = "SELECT * FROM students ORDER BY date"; break;
case 'student_id': $sql = "SELECT * FROM students ORDER BY student_id";
break;
case 'l_name': $sql = "SELECT * FROM students ORDER BY l_name"; break;

default: $sql = "SELECT * FROM students"; break;
}
if(isset($_POST["submit"])){
    $sql = "SELECT * FROM students WHERE gr=" . $_POST["gr"];
}

$result=mysql_query($sql); /* execute the query */
while($row=mysql_fetch_array($result)){
    echo "<h4> Name: " . $row["l_name"] . ', ' . $row["f_name"] . "</h4> \n";
    echo "<h5> ID: " . $row["student_id"] . "<br/> Email: " . $row["email"] .
    "<br/> Group: " . $row["gr"] . "<br/> Posted: " . $row["date"] . "</h5> \n";
}
mysql_close();
?>
</body>
</html>
```

[view the output page](#)

JavaScript Data Types & Variables

- JavaScript has only three primitive data types

```
String : "foo"  'how do you do?'  "I said 'hi'."      ""
Number: 12      3.14159      1.5E6
Boolean: true   false      *Find info on Null, Undefined
```

```
<html>
<!-- CS443 js02.html 16.08.06 -->

<head>
  <title>Data Types and Variables</title>
</head>

<body>
  <script type="text/javascript">
    var x, y;
    x= 1024;

    y=x; x = "foobar";
    document.write("<p>x = " + y + "</p>");
    document.write("<p>x = " + x + "</p>");
  </script>
</body>
</html>
```

[view page](#)

assignments are as in C++/Java

```
message = "howdy";
pi = 3.14159;
```

variable names are sequences of letters, digits, and underscores that start with a letter or an underscore

variables names are case-sensitive

you don't have to declare variables, will be created the first time used, but it's better if you use **var** statements

```
var message, pi=3.14159;
```

variables are loosely typed, can be assigned different types of values (Danger!)

JavaScript Operators & Control Statements

```
<html>
<!-- CS443 js03.html 08.10.10 -->

<head>
  <title>Folding Puzzle</title>
</head>

<body>
  <script type="text/javascript">
    var distanceToSun = 93.3e6*5280*12;
    var thickness = .002;

    var foldCount = 0;
    while (thickness < distanceToSun) {
      thickness *= 2;
      foldCount++;
    }
    document.write("Number of folds = " +
      foldCount);
  </script>
</body>
</html>
```

[view page](#)

standard C++/Java operators & control statements are provided in JavaScript

- +, -, *, /, %, ++, --, ...
- ==, !=, <, >, <=, >=
- &&, ||, !, ===, !==
- if, if-else, switch
- while, for, do-while, ...

PUZZLE: Suppose you took a piece of paper and folded it in half, then in half again, and so on.

How many folds before the thickness of the paper reaches from the earth to the sun?

*Lots of information is available online

JavaScript Math Routines

```
<html>
<!-- CS443 js04.html 08.10.10 -->

<head>
  <title>Random Dice Rolls</title>
</head>

<body>
  <div style="text-align:center">
    <script type="text/javascript">
      var roll1 = Math.floor(Math.random()*6) + 1;
      var roll2 = Math.floor(Math.random()*6) + 1;

      document.write("<img src='http://www.csc.liv.ac.uk/~martin/teaching/CS443/Images/die" +
        roll1 + ".gif' alt='dice showing ' + roll1 + '>");
      document.write("&nbsp;&nbsp;");
      document.write("<img src='http://www.csc.liv.ac.uk/~martin/teaching/CS443/Images/die" +
        roll2 + ".gif' alt='dice showing ' + roll2 + '>");
    </script>
  </div>
</body>
</html>
```

the built-in Math object contains functions and constants

```
Math.sqrt
Math.pow
Math.abs
Math.max
Math.min
Math.floor
Math.ceil
Math.round
```

```
Math.PI
Math.E
```

```
Math.random
function returns a real number in [0..1)
```

Interactive Pages Using Prompt

```
<html>
<!-- CS443 js05.html 08.10.10 -->

<head>
  <title>Interactive page</title>
</head>

<body>
<script type="text/javascript">
  var userName = prompt("What is your name?", "");
  var userAge = prompt("Your age?", "");
  var userAge = parseFloat(userAge);

  document.write("Hello " + userName + ".")
  if (userAge < 18) {
    document.write(" Do your parents know " +
      "you are online?");
  }
  else {
    document.write(" Welcome friend!");
  }
</script>

<p>The rest of the page...</p>
</body>
</html>
```

crude user interaction can take place using **prompt**

1st argument: the prompt message that appears in the dialog box

2nd argument: a default value that will appear in the box (in case the user enters nothing)

the function returns the value entered by the user in the dialog box (a string)

if value is a number, must use **parseFloat** (or **parseInt**) to convert

forms will provide a better interface for interaction
(later)

[view page](#)

User-Defined Functions

- function definitions are similar to C++/Java, except:
 - no return type for the function (since variables are loosely typed)
 - no variable typing for parameters (since variables are loosely typed)
 - by-value parameter passing only (parameter gets copy of argument)

```
function isPrime(n)
// Assumes: n > 0
// Returns: true if n is prime, else false
{
    if (n < 2) {
        return false;
    }
    else if (n == 2) {
        return true;
    }
    else {
        for (var i = 2; i <= Math.sqrt(n); i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

Can limit variable scope to the function.

if the first use of a variable is preceded with **var**, then that variable is local to the function

for modularity, should make all variables in a function local

JavaScript Objects

- an object defines a new type (formally, *Abstract Data Type*)
 - encapsulates data (properties) and operations on that data (methods)
- a String object encapsulates a sequence of characters, enclosed in quotes

properties include

`length` : stores the number of characters in the string

methods include

`charAt(index)` : returns the character stored at the given index (as in C++/Java, indices start at 0)

`substring(start, end)` : returns the part of the string between the start (inclusive) and end (exclusive) indices

`toUpperCase()` : returns copy of string with letters uppercase

`toLowerCase()` : returns copy of string with letters lowercase

to create a **string**, assign using `new` or (in this case) just make a direct assignment (`new` is implicit)

```
word = new String("foo");      word = "foo";
```

properties/methods are called exactly as in C++/Java

```
word.length          word.charAt(0)
```

JavaScript Arrays

- arrays store a sequence of items, accessible via an index
since JavaScript is loosely typed, elements do not have to be the same type
 - to create an array, allocate space using `new` (or can assign directly)

```
items = new Array(10);      // allocates space for 10 items
items = new Array();        // if no size given, will adjust dynamically
items = [0,0,0,0,0,0,0,0,0]; // can assign size & values []
```

- to access an array element, use `[]` (as in C++/Java)

```
for (i = 0; i < 10; i++) {
    items[i] = 0;           // stores 0 at each index
}
```

- the `length` property stores the number of items in the array

```
for (i = 0; i < items.length; i++) {
    document.write(items[i] + "<br>"); // displays elements
}
```

Arrays (cont.)

- Arrays have predefined methods that allow them to be used as stacks, queues, or other common programming data structures.

```
var stack = new Array();
stack.push("blue");
stack.push(12);           // stack is now the array ["blue", 12]
stack.push("green");     // stack = ["blue", 12, "green"]
var item = stack.pop();  // item is now equal to "green"
```

```
var q = [1,2,3,4,5,6,7,8,9,10];
item = q.shift();        // item is now equal to 1, remaining
                        // elements of q move down one position
                        // in the array, e.g. q[0] equals 2
q.unshift(125);         // q is now the array [125,2,3,4,5,6,7,8,9,10]
q.push(244);            // q = [125,2,3,4,5,6,7,8,9,10,244]
```

Date Object

- String & Array are the most commonly used objects in JavaScript
 - other, special purpose objects also exist
- the Date object can be used to access the date and time
 - to create a Date object, use new & supply year/month/day/... as desired

```
today = new Date();           // sets to current date & time  
  
newYear = new Date(2002,0,1); //sets to Jan 1, 2002 12:00AM
```

- methods include:

```
newYear.getYear()  
newYear.getMonth()  
newYear.getDay()  
newYear.getHours()  
newYear.getMinutes()  
newYear.getSeconds()  
newYear.getMilliseconds()
```

can access individual components of a date

Date Example

```
<html>  
<!-- CS443 js11.html 16.08.2006 --&gt;<br/>  
<head>  
    <title>Time page</title>  
</head>  
  
<body>  
    Time when page was loaded:  
    <script type="text/javascript">  
        now = new Date();  
  
        document.write("<p>" + now + "</p>");  
  
        time = "AM";  
        hours = now.getHours();  
        if (hours > 12) {  
            hours -= 12;  
            time = "PM"  
        }  
        else if (hours == 0) {  
            hours = 12;  
        }  
        document.write("<p>" + hours + ":" +  
                     now.getMinutes() + ":" +  
                     now.getSeconds() + " " +  
                     time + "</p>");  
  
</script>  
</body>  
</html>
```

by default, a date will be displayed in full, e.g.,

Sun Feb 03 22:55:20 GMT-0600
(Central Standard Time) 2002

can pull out portions of the date using the methods and display as desired

here, determine if "AM" or "PM" and adjust so hour between 1-12

10:55:20 PM

[view page](#)

Another Example

```
<html>  
<!-- CS443 js12.html 12.10.2012 --&gt;<br/>  
<head>  
    <title>Time page</title>  
</head>  
  
<body>  
    <p>Elapsed time in this year:  
    <script type="text/javascript">  
        now = new Date();  
        newYear = new Date(2012,0,1);  
  
        secs = Math.round((now-newYear)/1000);  
  
        days = Math.floor(secs / 86400);  
        secs -= days*86400;  
        hours = Math.floor(secs / 3600);  
        secs -= hours*3600;  
        minutes = Math.floor(secs / 60);  
        secs -= minutes*60;  
  
        document.write(days + " days, " +  
                      hours + " hours, " +  
                      minutes + " minutes, and " +  
                      secs + " seconds.");  
    </script>  
    </p>  
</body>  
</html>
```

you can add and subtract Dates:
the result is a number of milliseconds

here, determine the number of seconds since New Year's day
(note: January is month 0)

divide into number of days, hours, minutes and seconds

[view page](#)

Document Object

Internet Explorer, Firefox, Opera, etc. allow you to access information about an HTML document using the `document` object

```
<html>  
<!-- CS443 js13.html 2.10.2012 --&gt;<br/>  
<head>  
    <title>Documentation page</title>  
</head>  
  
<body>  
    <table width="100%">  
        <tr>  
            <td><i>  
                <script type="text/javascript">  
                    document.write(document.URL);  
                </script>  
            </i></td>  
            <td style="text-align: right;"><i>  
                <script type="text/javascript">  
                    document.write(document.lastModified);  
                </script>  
            </i></td>  
        </tr>  
    </table>  
</body>  
</html>
```

`document.write(...)`
method that displays text in the page

`document.URL`
property that gives the location of the HTML document

`document.lastModified`
property that gives the date & time the HTML document was last changed

[view page](#)

User-Defined Objects

- can define new objects, but the notation can be somewhat awkward
 - simply define a function that serves as a constructor
 - specify data fields & methods using `this`
- no data hiding: can't protect data or methods

```
// CS443      Die.js      11.10.2011 //
// Die class definition
///////////////////////////////



function Die(sides)
{
    this.numSides = sides;
    this.numRolls = 0;
    this.roll = roll; // define a pointer to a function
}

function roll()
{
    this.numRolls++;
    return Math.floor(Math.random() * this.numSides) + 1;
}
```

define `Die` function (i.e., the object's constructor)
initialize data fields in the function, preceded with "this"
similarly, assign method to separately defined function (which uses `this` to access data)

Object Example

```
<html>
<!-- CS443 js15.html 11.10.2011 -->

<head>
<title>Dice page</title>

<script type="text/javascript"
       src="Die.js">
</script>
</head>

<body>
<script type="text/javascript">
    die6 = new Die(6);
    die8 = new Die(8);

    roll16 = -1; // dummy value to start loop
    roll8 = -2; // dummy value to start loop
    while (roll16 != roll18) {
        roll16 = die6.roll();
        roll18 = die8.roll();

        document.write("6-sided: " + roll16 +
                      "&nbsp;&nbsp;&nbsp;" +
                      "8-sided: " + roll18 + "<br />");

        document.write("<br />Number of rolls: " +
                      die6.numRolls);
    }
</script>
</body>
</html>
```

create a `Die` object using `new` (similar to `String` and `Array`)

here, the argument to `Die` initializes `numSides` for that particular object

each `Die` object has its own properties (`numSides` & `numRolls`)

`Roll()`, when called on a particular `Die`, accesses its `numSides` property and updates its `NumRolls`

[view page](#)

JavaScript and HTML validators

- In order to use an HTML validator, and not get error messages from the JavaScript portions, you must "mark" the JavaScript sections in a particular manner. Otherwise the validator will try to interpret the script as HTML code.

- To do this, you can use a markup like the following in your inline code (this isn't necessary for scripts stored in external files).

```
<script type="text/javascript">
    // <![CDATA[
        document.write("<p>The quick brown fox jumped over the lazy dogs.</p>");
        // **more code here, etc.
    // ]]>
</script>
```

- Since the (new) XHTML standard is written as an XML application, validators such as the one from the W3C are actually attempting to check an XML document for the correct structure.

- The two tags `<![CDATA[` and `]]>` together form an XML directive, meaning to interpret the data between them as literal (non-parsed) "character data". An XML validator will effectively ignore the data between these two tags, meaning that any symbols that would result in an invalid document structure are ignored and do not result in an error message from the validator.

- Because we are using these tags inside of a JavaScript block, and they are not JavaScript commands, we precede each of them with a (JavaScript) comment marker, hence the two forward slashes before each tag.

Overview

The Document Object Model ([DOM](#)) is an API that allows programs to interact with HTML (or XML) documents

- In typical browsers, the JavaScript version of the API is provided
- W3C recommendations define standard DOM

DOM Level 1:

W3C recommendation, 1 Oct. 1998.

Interfaces for representing an XML and HTML document.

- 1)**Document**
- 2)**Node**
- 3)**Attr**
- 4)**Element**
- 5)**and Text interfaces.**

DOM Level 2:

W3C recommendation, 13 Nov. 2000.

It contains six different specifications:

- 1)DOM2 Core
- 2)Views
- 3)Events
- 4)Style
- 5)Traversal and Range
- 6)and the DOM2 HTML.

DOM Level 3:

W3C candidate recommendation, 7 Nov. 2003

It contains five different specifications:

- 1)DOM3 Core
- 2)Load and Save
- 3)Validation
- 4)Events
- 5)and XPath

DOM Introduction

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html>
<head>
<title>Rollover.html</title>
<script type="text/javascript" src="rollover.js">
</script>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<p>

</body>
</html>
```

19

DOM Introduction

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html>
<head>
<title>Rollover.html</title>
<script type="text/javascript" src="rollover.js">
</script>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<p>

</body>
</html>
```

20

DOM Introduction

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html>
<head>
<title>Rollover.html</title>
<script type="text/javascript" src="rollover.js">
</script>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body> Default language for scripts specified as attribute values
<p>

</body>
</html>
```

21

DOM Introduction

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html>
<head>
<title>Rollover.html</title>
<script type="text/javascript" src="rollover.js">
</script>
<meta http-equiv="Content-Script-Type" content="text/javascript" />
</head>
<body>
<p>

</body>
</html>
```

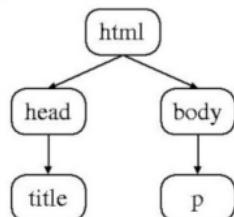
22

20

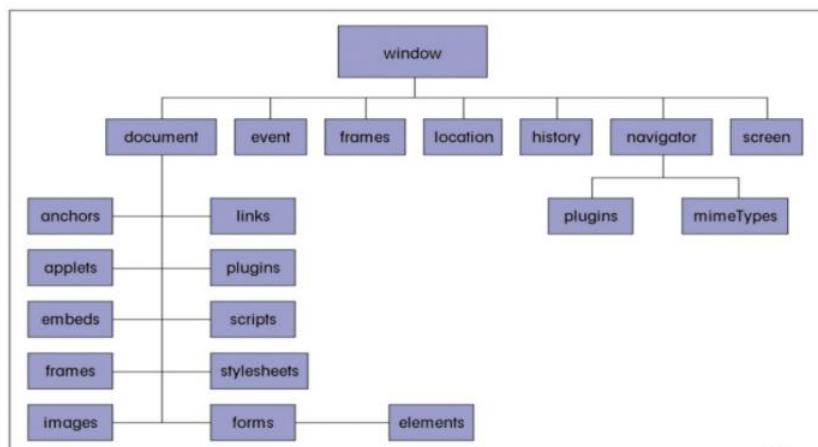
Document Tree

Recall that HTML document elements form a tree structure, e.g.,

DOM allows scripts to **access** and **modify** the document tree



The Document Tree Example



31

Referencing Objects-Each Object is Identified by Object Name.

Object Name	Description
window	The browser window
document	The Web document displayed in the window
document.body	The body of the Web document displayed in the browser window
event	Events or actions occurring within the browser window
history	The list of previously visited Web sites within the browser window
location	The URL of the document currently displayed in the browser window
navigator	The browser itself
screen	The screen displaying the document

How To Use Referencing Objects

Object Names

- General form is `object1.object2.object3..`

TO Access The History

`window.history`

33

To Access The Body

`document.body`

Document Tree: Node

```
<body>                                Example HTML document
  <p>
    Text within a "p" element.
  </p>
  <ol>
    <li>First element of ordered list.</li>
    <li>Second element.</li>
  </ol>
  <!-- Call function producing an outline of this document's
      element tree -->
  <form action="">
    <p><input type="button" name="button" value="Click to see outline"
           onclick="window.alert(<treeOutline()>);" /></p>
  </form>
</body>                                Function will use Node methods and
                                         properties to produce string
                                         representing Element tree
```

An Example of DOM XML

```
<?xml version = "1.0"?>
<message from = "Paul" to = "Name">
  <body>Hello!</body>
</message>
```

Node is created for **message** element:

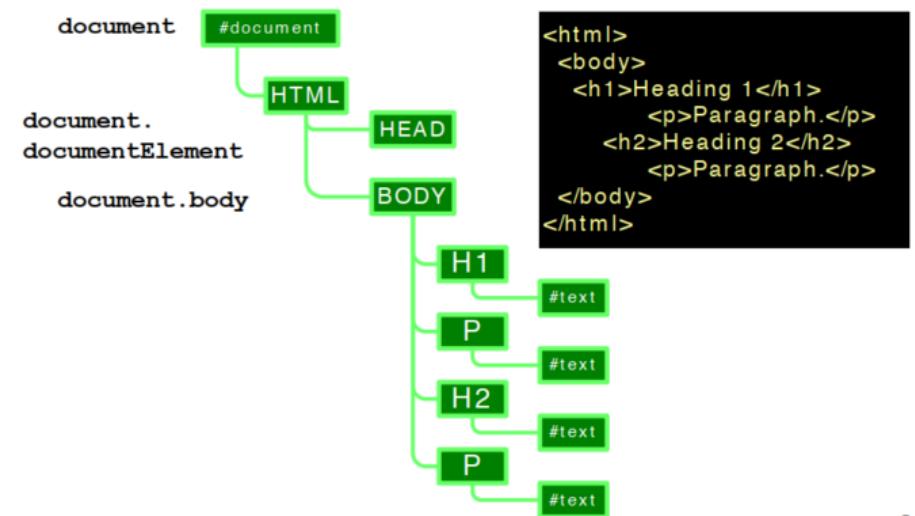
- **message** element has child element: **body**.
- **body** element has Text "Hello!"
- Attributes: **from** and **to** also are nodes in DOM tree.

The DOM Interface

- The DOM has many interfaces to handle various node objects.
- Every interface has its "Attributes" and "Methods".
 - Compare with Object Oriented Programming (OOP).

DOM	OOP
Interface	Object Class
Attribute	Property
Method	Method

Document Tree Structure



DOM NODE Methods

Method Name	Description
<code>appendChild</code>	Appends a child node.
<code>cloneNode</code>	Duplicates the node.
<code>getAttributes</code>	Returns the node's attributes.
<code>getChildNodes</code>	Returns the node's child nodes.
<code>getNodeName</code>	Returns the node's name.
<code>getNodeType</code>	Returns the node's type (e.g., element, attribute, text, etc.).
<code>getNodeValue</code>	Returns the node's value.
<code>getParentNode</code>	Returns the node's parent.
<code>hasChildNodes</code>	Returns <code>true</code> if the node has child nodes.
<code>removeChild</code>	Removes a child node from the node.
<code>replaceChild</code>	Replaces a child node with another node.
<code>setnodeValue</code>	Sets the node's value.
<code>insertBefore</code>	Appends a child node in front of a child node.

42

NAVIGATOR :-Some properties are **read-only**

Browser Property	Description
<code>navigator.appName</code>	The name of the browser
<code>navigator.appVersion</code>	The major version number of the browser (may also include a compatibility value and the name of the operating system)
<code>navigator.appMinorVersion</code>	The minor version number of the browser
<code>navigator.appCodeName</code>	The name of the browser's code
<code>navigator.userAgent</code>	The name of the browser associated user agent
<code>navigator.platform</code>	The operating system under which the browser is running
<code>navigator.cpuClass</code>	The type of CPU in use with the browser
<code>navigator.systemLanguage</code>	The language used by the browser
<code>navigator.cookieEnabled</code>	A Boolean value indicating whether cookies are enabled

Example For NAVIGATOR

```
<html><body><script type="text/javascript">
document.write("<p>Browser: ");
document.write(navigator.appName + "</p>");
document.write("<p>Browserversion: ");
document.write(navigator.appVersion + "</p>");
document.write("<p>Code: ");
document.write(navigator.appCodeName + "</p>");
document.write("<p>Platform: ");
document.write(navigator.platform + "</p>");
document.write("<p>Cookies enabled: ");
document.write(navigator.cookieEnabled + "</p>");
document.write("<p>Browser's user agent header: ");
document.write(navigator.userAgent + "</p>");
</script></body></html>
```

51

DOM Event Handling

JavaScript [event listener](#): function that is called with Event instance when a certain event occurs

An event listener is associated with a target element by calling `addEventListener()` on the element

```
var button = window.document.getElementById("msgButton");
button.addEventListener("click", sayHello, false);

function sayHello(event) {
  window.alert(
    "Hello World!\n\n" +
    "Event type: " + event.type + "\n" +
    "Event target element type: " + event.target.nodeName);
  return;
}
```

```

var button = window.document.getElementById("msgButton");
button.addEventListener("click", sayHello, false);

```

Event handler

```

function sayHello(event) {
    window.alert(
        "Hello World!\n\n" +
        "Event type: " + event.type + "\n" +
        "Event target element type: " + event.target.nodeName);
    return;
}

```



```

var button = window.document.getElementById("msgButton");
button.addEventListener("click", sayHello, false);

```

Event instance

```

function sayHello(event) {
    window.alert(
        "Hello World!\n\n" +
        "Event type: " + event.type + "\n" +
        "Event target element type: " + event.target.nodeName);
    return;
}

```



```

var button = window.document.getElementById("msgButton");
button.addEventListener("click", sayHello, false);

```

Normally false
(more later)

```

function sayHello(event) {
    window.alert(
        "Hello World!\n\n" +
        "Event type: " + event.type + "\n" +
        "Event target element type: " + event.target.nodeName);
    return;
}

```

Things change...

- Until recently, we didn't have any alternative to this load/wait/respond method of web browsing.
- Ajax (sometimes written AJAX) is a means of using JavaScript to communicate with a web server without submitting a form or loading a new page.
- Ajax makes use of a built-in object, XMLHttpRequest, to perform this function.
- This object is not yet part of the DOM (Document Object Model) standard, but is supported (in different fashions) by Firefox, Internet Explorer, Safari, Opera, and other popular browsers.
- The term "Ajax" was coined in 2005, but the XMLHttpRequest object was first supported by Internet Explorer several years before this.

The XMLHttpRequest object (cont.)

```

function getXMLHttpRequest()
/* This function attempts to get an Ajax request object by trying
   a few different methods for different browsers. */
{
    var request, err;
    try {
        request = new XMLHttpRequest(); // Firefox, Safari, Opera, etc.
    }
    catch(err) {
        try { // first attempt for Internet Explorer
            request = new ActiveXObject("MSXML2.XMLHttp.6.0");
        }
        catch (err) {
            try { // second attempt for Internet Explorer
                request = new ActiveXObject("MSXML2.XMLHttp.3.0");
            }
            catch (err) {
                request = false; // oops, can't create one!
            }
        }
    }
    return request;
}

```

If this function doesn't return "false" then we were successful in creating an XMLHttpRequest object.

XMLHttpRequest object properties

Property	Description
readyState	An integer from 0 . . . 4. (0 means the call is uninitialized, 4 means that the call is complete)
onreadystatechange	Determines the function called when the objects readyState changes.
responseText	Data returned from the server as a text string (read-only).
responseXML	Data returned from the server as an XML document object (read-only).
status	HTTP status code returned by the server
statusText	HTTP status phrase returned by the server

We use the `readyState` to determine when the request has been completed, and then check the `status` to see if it executed without an error. (We'll see how to do this shortly.)

XMLHttpRequest object methods

Method	Description
open('method', 'URL', asyn)	Specifies the HTTP method to be used (GET or POST as a string, the target URL, and whether or not the request should be handled asynchronously (asyn should be true or false, if omitted, true is assumed).
send(content)	Sends the data for a POST request and starts the request, if GET is used you should call <code>send(null)</code> .
setRequestHeader('x', 'y')	Sets a parameter and value pair <code>x=y</code> and assigns it to the header to be sent with the request.
getAllResponseHeaders()	Returns all headers as a string.
getResponseHeader(x)	Returns header <code>x</code> as a string.
abort()	Stops the current operation.

The `open` object method is used to set up the request, and the `send` method starts the request by sending it to the server (with data for the server if the POST method is used).

A general skeleton for an Ajax application

```
<script type="text/javascript">
    // ***** include the getXMLHttpRequest function defined before
    var ajaxRequest = getXMLHttpRequest();

    if (ajaxRequest) {    // if the object was created successfully

        ajaxRequest.onreadystatechange = ajaxResponse;
        ajaxRequest.open("GET", "search.php?query=Bob");
        ajaxRequest.send(null);
    }

    function ajaxResponse()    //This gets called when the readyState changes.
    {
        if (ajaxRequest.readyState != 4)    // check to see if we're done
            { return; }
        else {
            if (ajaxRequest.status == 200) // check to see if successful
                {    // process server data here. . . }
            else {
                alert("Request failed: " + ajaxRequest.statusText);
            }
        }
    }
</script>
```

The other PHP scripts for the time examples

- Here's the script with a simple HTML tag in it.

```
<?php
echo '<span style="color: red;">' . date('H:i:s') . "</span>";
?>
```

- The output with a table.

```
<?php
$tr = '<tr style="border: 2px solid;">';
$td = '<td style="border: 2px solid">';

$table = '<table style="border: 2px solid; margin: auto;">';
$table .= $tr . $td . date('j M Y') . '</td></tr>';
$table .= $tr . $td . date('H:i:s') . '</td></tr>';
$table .= '</table>';
echo $table;
?>
```

Accessing an XML document in JavaScript

- To use an XML document in JavaScript, you must create an object to hold it. This can be done in the following fashion:
- Non-Microsoft browsers:

```
<script>
    var myXMLDoc = document.implementation.createDocument("", "", null);
    myXMLDoc.load("mydoc.xml");
    // other code here
</script>
```

- Internet Explorer:

```
<script>
    var myXMLDoc = new ActiveXObject("Microsoft.XMLDOM");
    myXMLDoc.async="false";
    myXMLDoc.load("mydoc.xml");
    // other code here
</script>
```

- Once we've created the object holding the XML document, we can then use JavaScript methods to examine it, extract data from it, etc.

The “time” example using XML

- The first change is to make a new PHP script that returns an XML document to the browser.

```
<?php
header('Content-Type: text/xml');
echo "<?xml version=\"1.0\" ?>\n";
echo "<clock><timenow>" . date('H:i:s') . "</timenow></clock>";
?>
```

- After that change (and inserting the new script name into the HTML code), we need to alter the ajaxResponse function to parse the XML document. That new JavaScript function is given on the next slide.
- Note that we need not explicitly create an object to hold the XML document, but that responseXML (as a property of XMLHttpRequest) is already such an object.

The new Ajax response function

```
function ajaxResponse() //This gets called when the readyState changes.
{
    if (ajaxRequest.readyState != 4) // check to see if we're done
        { return; }
    else {
        if (ajaxRequest.status == 200) // check to see if successful
            { var timeValue =
                ajaxRequest.responseXML.getElementsByTagName("timenow") [0];
                document.getElementById("showtime").innerHTML =
                    timeValue.childNodes[0].nodeValue; }

        else {
            alert("Request failed: " + ajaxRequest.statusText);
            }
    }
}
```

- This new response function uses a JavaScript method to access the XML DOM and retrieve the time string before inserting it into the output display box.

```
<?php
header("Content-Type: text/xml");
echo "<?xml version=\"1.0\" ?>\n";
echo "<names>\n";
if(!$query) $query = strtoupper($_GET['query']);

if($query != "")
{
    include_once('db_access.php');
    $connection = mysql_connect($db_host, $db_username, $db_password);
    if(!$connection)
    {
        exit("Could not connect to the database: <br/>" . htmlspecialchars(mysql_error()));
    }
    mysql_select_db($db_database);

    $select = "SELECT ";
    $column = "name ";
    $from = "FROM ";
    $tables = "people ";
    $where = "WHERE ";
    $condition = "upper(name) LIKE '%$query%'";
    $SQL_query = htmlentities($select . $column . $from . $tables . $where . $condition);
    $result = mysql_query($SQL_query);
    while ($row = mysql_fetch_row($result))
    {
        echo "<name>" . $row[0] . "</name>\n";
    }
    mysql_close($connection);
}
echo "</names>";
?>
```

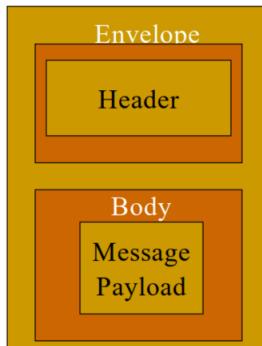
[view the output page](#)

Three Most Common Styles of Use

- **RPC** (Remote Procedure Calls)
 - A distributed function call interface
- **SOAP** (Simple Object Access Protocol)
 - The basic unit of communication is a message, rather than an operation
- **REST** (Representational State Transfer)
 - Standard operations in HTTP: GET, POST, PUT, DELETE
 - Interacting with stateful resources, rather than messages or operations

SOAP Web Services

- Basic unit: message
- Supported by most major vendors, loose coupling



Representational State Transfer (REST)

- Interacting with **stateful resources**, rather than messages or operations
- Using HTTP standard operations such as GET, POST, PUT, DELETE
- WSDL 2.0 offers support for binding to all HTTP request methods
 - WSDL 1.1 only GET and POST

SOAP web service

- HTTP-XML-based protocol
- Enables application to communicate over Internet
- Uses XML documents called messages
- SOAP message contains an envelope
- Describes message's content and intended recipient
- Ability to make a Remote Procedure Call (RPC)
- Request to another machine to run a task

SOAP

- Using Web Services and SOAP, the request would look something like this:

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:body pb="http://www.acme.com/phonebook">
        <pb:GetUserDetails>
            <pb:UserID>12345</pb:UserID>
        </pb:GetUserDetails>
    </soap:Body>
</soap:Envelope>
```

Pros & cons

Advantages

- Human readable XML
- Easy to debug
- SOAP runs over HTTP
- Firewalls not affected
- Services can be written in any language, platform or operating system

Disadvantages

- S-L-O.....-W
- XML produces a lot of overhead for small messages
- Web Services speed relies on Internet traffic conditions
- Not strictly-typed XML

So what's REST already?

- REpresentational State Transfer
- An architectural style, not a toolkit
- “We don't need no toolkits!”
- A distillation of the way the Web already works

REST defined

- Resources are identified by uniform resource identifiers (URIs)
- Resources are manipulated through their representations
- Messages are self-descriptive and stateless
- Multiple representations are accepted or sent
- Hypertext is the engine of application state

HTTP Request/Response As REST



15

REST style

- Client-server
- Stateless
- Cached
- Uniform interface
- Layered system
- (Code on demand)

State

- “State” means application/session state
- Maintained as part of the content transferred from client to server back to client
- Thus any server can potentially continue transaction from the point where it was left off
- State is never left in limbo

Transfer of state

- Connectors (client, server, cache, resolver, tunnel) are unrelated to sessions
- State is maintained by being transferred from clients to servers and back to clients

REST and HTTP

- REST is a *post hoc* description of the Web
- HTTP 1.1 was designed to conform to REST
- Its methods are defined well enough to get work done
- Unsurprisingly, HTTP is the most RESTful protocol
- But it's possible to apply REST concepts to other protocols and systems

What do REST messages look like?

- Like what we already know: HTTP, URLs, etc.
- REST can support any media type, but XML is expected to be the most popular transport for structured information.
- Unlike SOAP and XML-RPC, REST does not really require a new message format

SOAP vs. REST

- Using Web Services and SOAP, the request would look something like this:

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:body pb="http://www.acme.com/phonebook">
        <pb: GetUserDetails>
            <pb: UserID>12345</pb: UserID>
        </pb: GetUserDetails>
    </soap:Body>
</soap:Envelope>
```

SOAP vs. REST

SOAP	REST	
Meaning	Simple Object Access Protocol	Representational State Transfer
Design	Standardized protocol with pre-defined rules to follow.	Architectural style with loose guidelines and recommendations.
Approach	Function-driven (data available as services, e.g.: "getUser")	Data-driven (data available as resources, e.g. "user").
Statefulness	Stateless by default, but it's possible to make a SOAP API stateful.	Stateless (no server-side sessions).
Caching	API calls cannot be cached.	API calls can be cached.
Security	WS-Security with SSL support. Built-in ACID compliance.	Supports HTTPS and SSL.
Performance	Requires more bandwidth and computing power.	Requires fewer resources.
Message format	Only XML.	Plain text, HTML, XML, JSON, YAML, and others.
Transfer protocol(s)	HTTP, SMTP, UDP, and others.	Only HTTP
Recommended for	Enterprise apps, high-security apps, distributed environment, financial services, payment gateways, telecommunication services.	Public APIs for web services, mobile services, social networks.
Advantages	High security, standardized, extensibility.	Scalability, better performance, browser-friendliness, flexibility.
Disadvantages	Poorer performance, more complexity, less flexibility.	Less security, not suitable for distributed environments.

Giới thiệu XML

XML: eXtensible Markup Language - là một ngôn ngữ đánh dấu được sử dụng để tạo ra thẻ riêng, tạo nên các văn bản với dữ liệu tự mô tả.

Được tạo nên bởi Liên minh mạng toàn cầu W3Schools nhằm khắc phục những hạn chế của HTML - ngôn ngữ đánh dấu siêu văn bản. Giống như HTML, XML cũng được dựa trên SGML – Standard Generalized Markup Language.

Là cơ sở của nền công nghệ thương mại điện tử, các công ty đang sử dụng XML để giải quyết những vấn đề kinh doanh.

HTML	XML
HTML được thiết kế cho mục đích trình bày dữ liệu	XML được thiết kế cho mục đích lưu trữ và truyền tải dữ liệu giữa các hệ thống khác nhau
HTML dùng để hiển thị dữ liệu và chú trọng vào việc dữ liệu được hiển thị như thế nào	XML dùng để mô tả dữ liệu và chú trọng vào nội dung của dữ liệu
HTML hiển thị thông tin	XML mô tả thông tin

XML cung cấp một phương tiện dùng văn bản (*text*) để mô tả thông tin và áp dụng một cấu trúc kiểu cây cho thông tin đó.

Tại mức căn bản, mọi thông tin đều thể hiện dưới dạng text, chen giữa là các thẻ đánh dấu (*markup*) với nhiệm vụ ký hiệu sự phân chia thông tin thành một cấu trúc có thứ bậc, các phần tử (*element*) dùng để chứa dữ liệu và các thuộc tính của các phần tử đó.

Văn bản có cấu trúc XML cho phép biểu diễn thông tin về các đối tượng trong thực tế

XML dùng để phục vụ cho việc mô tả dữ liệu (*thông tin lưu trữ bao gồm những gì, lưu trữ ra sao*) để các hệ thống khác nhau có thể đọc và sử dụng những thông tin này một cách thuận tiện

Các thẻ (*tag*) của XML thường không được định nghĩa trước mà chúng được tạo ra theo quy ước của người, (hoặc *Chương trình*) tạo ra XML theo những quy ước riêng.

XML sử dụng các khai báo kiểu dữ liệu DTD (Document Type Definition) hay lược đồ Schema để mô tả dữ liệu.

Ưu điểm XML

Dữ liệu độc lập là ưu điểm chính của XML. Do XML chỉ dùng để mô tả dữ liệu bằng dạng text nên tất cả các chương trình đều có thể đọc được XML.

Dễ dàng đọc và phân tích dữ liệu, nhờ ưu điểm này mà XML thường được dùng để trao đổi dữ liệu giữa các hệ thống khác nhau.

Dễ dàng tạo 1 file XML.

Lưu trữ cấu hình cho web site

Sử dụng cho phương thức Remote Procedure Calls (RPC) phục vụ web service

XML sử dụng bộ ký tự toàn cầu Universal Character Set làm cơ sở, kết hợp các chuỗi ký tự với nhau tạo nên một tài liệu XML.

XML dùng để mô tả thông tin nhưng không biết về ngữ nghĩa của dữ liệu. Vậy nên được dùng cho nhiều loại dữ liệu đa phương tiện.

Rất nhiều các phần mềm soạn thảo hỗ trợ soạn thảo và bảo trì XML.

Dữ liệu có tên, cấu trúc thứ bậc và các thuộc tính.

XML có cú pháp chung cho các tài liệu để các phần mềm XML Parser có thể đọc và phân tích, hiểu bối cục tương đối của thông tin trong tài liệu.

XML không hạn chế về việc được sử dụng như thế nào, có rất nhiều các phần mềm với chức năng trừu tượng hóa dữ liệu thành các định dạng khác giàu thông tin hơn.

Ứng dụng của XML

Mô tả cấu hình của 1 Website, ứng dụng. Ví dụ trong ASP.NET là tập tin `web.config`; khi xây dựng web application bằng JSP là `faces-config.xml` và `web.xml`.

Cung cấp tin, dữ liệu cho các hệ thống khác nhau để có thể khai thác, sử dụng. Ví dụ sử dụng tính năng cung cấp RSS của các web site có cung cấp tính năng dạng này như : www.vnExpress.net, www.tuoitre.vn, ... để lấy tin tự động như giá vàng, tin thể thao, thời sự, tin thời tiết ...

Xây dựng các hệ thống thu thập dữ liệu XML theo thời gian từ các hệ thống con.

Cú pháp XML

Văn bản có cấu trúc XML cho phép biểu diễn thông tin về các đối tượng trong thực tế.

Đối tượng x thuộc loại X trong thực tế được biểu diễn bởi thẻ X trong tài liệu XML bao gồm cả các thuộc tính a của x.

Ví dụ:

Phân số 4/5 trong thực tế có thẻ:

```
<PHAN_SO Tu_so="4" Mau_so="5" />
```

Dãy các số nguyên a bao gồm các số nguyên 1,4,5,-3 sẽ được biểu diễn bởi thẻ

```
<DAY_SO>
    <SO Gia_tri="1" />
    <SO Gia_tri="4" />
    <SO Gia_tri="5" />
    <SO Gia_tri="-3" />
</DAY_SO>
```

Hệ thống các thẻ đánh dấu:

Các thẻ đánh dấu trong ngôn ngữ theo định chuẩn XML bao gồm 2 loại: Thẻ có nội dung và thẻ rỗng.

- Các thẻ có nội dung có dạng:

```
<Tên> Nội dung </Tên>
```

- Các thẻ rỗng có dạng:

```
<Tên />
```

Các thẻ có thể có hoặc không có các thuộc tính (trong cùng thẻ). Thuộc tính trong một thẻ có dạng:

```
Ten_thuoc_tinh="Gia_tri"
```

Định chuẩn XML

Ví dụ : Tài liệu XML

```
<?xml version="1.0" encoding="utf-8"?>
<DUONG_TRON Ban_kinh="5">
<DIEM x="4" y="2"/>
</DUONG_TRON>
```

Định chuẩn XML

Quan hệ lồng nhau giữa các thẻ có nội dung:

- Nội dung bên trong thẻ có nội dung có thể là các thẻ khác. Khi thẻ A có nội dung là thẻ B ta gọi: Thẻ A là thẻ cha của B , thẻ A chứa thẻ B.
- Qui định yêu cầu các thẻ với quan hệ lồng nhau hoàn toàn. Khi thẻ A là thẻ cha của thẻ B, A phải chứa phần bắt đầu và cả phần kết thúc của thẻ B.

Thẻ A là thẻ cha của B với dạng lồng nhau hoàn toàn (hợp lệ):

```
<A>
  <B> ....</B>
</A>
```

Thẻ A là thẻ cha của B với dạng lồng nhau không hoàn toàn (không hợp lệ):

```
<A>
  <B ></A>
</B>
```

Một tài liệu XML phải có duy nhất một và chỉ một thẻ chứa tất cả các thẻ còn lại, gọi là **thẻ gốc – Root element (Document element)**:

Ví dụ:

```
<?xml version="1.0"?>
<Blog>
  <Title>Let me know what you think
  </Title>
  <Author>Yin Yang</Author>
</Blog>
```

Các kiểu tài liệu XML:

- Well-formed Document**: tài liệu XML đúng cú pháp.
- DTD - Constrained Document**: Tạo XML có khai báo DTD (*Document type definition*) để mô tả cấu trúc dữ liệu trong XML.
- XML-Schema - Constrained Document**: Tạo XML có sử dụng “*lược đồ*” Schema để kiểm tra tính hợp lệ của XML.

Well-formed XML Document (đúng cú pháp) :

- Có duy nhất có một phần tử thuộc cấp cao nhất trong tài liệu, còn gọi là nút gốc (*root element*).
- Mỗi **một thẻ mở đều phải có thẻ đóng** và tên thẻ là phân biệt hoa thường.
- Các thẻ khi đóng phải theo **đúng trình tự** (*mở sau đóng trước*)
- Tên thẻ không nên có khoảng trắng, không nên bắt đầu bằng “xml”.
- Các thuộc tính (*atributes*) của một thẻ luôn luôn tồn tại theo cặp theo quy ước: **<Tên> = <giá trị>**; không nên đặt tên thuộc tính trùng nhau, và giá trị của thuộc tính phải đặt trong cặp dấu nháy kép hay nháy đơn. Tên của thuộc tính (*attribute*) sẽ theo qui luật đặt tên giống như đối với tên thẻ.
- Các thẻ (tag) trong XML có thể **lồng nhau** (*Thẻ này có thể chứa nhiều thẻ khác ở bên trong*).

Nội dung của tài liệu XML bao gồm 2 phần:

- **Nội dung chính:** Hệ thống các thẻ đánh dấu (có hoặc không có nội dung) tương ứng với các thông tin cần biểu diễn.
- **Nội dung phụ:** Hệ thống các thẻ khác có ý nghĩa bổ sung, tăng cường một số thông tin về tài liệu XML. Các thẻ này có tác dụng giúp cho việc sử dụng, xử lý trên tài liệu XML tốt hơn trong một số trường hợp nhất định.

Các thẻ bên trong nội dung **phụ** bao gồm:

- Thẻ khai báo tham số
- Thẻ chỉ thị xử lý
- Thẻ ghi chú
- Thẻ CDATA
- Thẻ khai báo cấu trúc (đặc tả cấu trúc với DTD)
- Thẻ khai báo thực thể (Kĩ thuật đặc tả nội dung tài liệu XML)

Thẻ khai báo tham số

Thẻ khai báo tham số: mô tả thêm một số thông tin chung (tham số) về tài liệu XML ngoài các thông tin biểu diễn trong nội dung chính.

Cú pháp:

```
<?xml Ten_1="Gia_tri_1" Ten_2="Gia_tri_2" ... ?>
```

Ten_1, Ten_2, ...: các tên các tham số và Gia_tri_1, Gia_tri_2, ... là các giá trị tương ứng. Có 3 tham số được dùng là version, encoding, và standalone.

Tham số **version**: Khai báo về phiên bản của định chuẩn XML được sử dụng.

Ví dụ :

Tài liệu XML thuộc định chuẩn 1.0

```
<?xml version="1.0" ?>
```

Tham số **encoding**: Khai báo về cách mã hóa các ký tự trong tài liệu.

Ví dụ: Tài liệu XML sử dụng cách mã hóa Unicode ký hiệu utf-8:

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

Tài liệu XML sử dụng cách mã hóa Unicode ký hiệu utf-16:

```
<?xml version="1.0" encoding="utf-16" ?>
```

Tham số **standalone**: Khai báo về liên kết của tài liệu XML và các tài liệu khác.

Tham số này chỉ có 2 giá trị hợp lệ là "yes", "no". Giá trị định sẵn là "no".

Ví dụ: Tài liệu XML có liên kết với các tài liệu khác:

```
<?xml standalone="yes" ?>
```

Tài liệu XML không có liên kết với các tài liệu khác:

```
<?xml version="1.0" standalone="no" ?>
```

Thẻ chỉ thị xử lý

Thẻ chỉ thị xử lý: cho phép mô tả thêm một số thông tin (liên quan xử lý) về tài liệu XML có ý nghĩa riêng với một công cụ xử lý nào đó.

Dạng khai báo chung:

```
<? Bo_xu_ly Du_lieu ?>
```

Bo_xu_ly là ký hiệu của bộ xử lý sẽ tiến hành một số xử lý nào đó trên tài liệu XML. Du_lieu là thông tin được gởi đến Bo_xu_ly.

Ví dụ:

```
<?xmlstylesheet type="text/css" href="Dinh_dang.css" ?>
```

- Là thẻ chỉ thị cần xử lý định dạng thể hiện tài liệu XML với “chương trình định dạng” theo ngôn ngữ CSS được lưu trữ bên trong tập tin Dinh_dang.css.

Thẻ ghi chú

Thẻ ghi chú: cho phép bổ sung các thông tin ghi chú có ý nghĩa đối với con người và hoàn toàn không có ý nghĩa với các hệ thống xử lý tài liệu XML.

Cú pháp:

```
<-- Nội dung ghi chú -->
```

Chú ý:

- Trong nội dung của ghi chú không có ký tự “-“.
- Không nên đặt ghi chú trong 1 thẻ (*Thuộc giới hạn mở thẻ ... đóng thẻ*).
- Không nên đặt ghi chú trước dòng khai báo `<?xml...?>`.

Thẻ CDATA

Thẻ CDATA: yêu cầu các bộ phân tích tài liệu XML Parser bỏ qua và không phân tích vào nội dung bên trong của thẻ này.

Tác dụng của thẻ là cho phép sử dụng trực tiếp bên trong thẻ một số ký hiệu không được phép nếu sử dụng bên ngoài (ví dụ các ký tự “<”, “>”, ...).

Dạng khai báo chung:

```
<![CDATA [ Nội dung ]]>
```

Thẻ PCDATA

PCDATA (*Parsed character data*): là dữ liệu sẽ được đọc và phân tích bởi chương trình phân tích XML.

Trong PCDATA không được phép dùng các ký tự đặc biệt có liên quan đến việc xác định các thành tố của XML như <,>,&, ...

Cấu trúc tài liệu XML

Khái niệm về **cấu trúc tài liệu XML**:

- Tương ứng với cấu trúc của nội dung chính
- Cách thức tổ chức, sắp xếp của các thẻ (có hay không có nội dung) trong nội dung chính.

Ngôn ngữ đặc tả cấu trúc: Có rất nhiều ngôn ngữ đặc tả để mô tả cấu trúc tài liệu XML như: DTD, XML Schema, XMI- Data, Schematron, RELAX NG, v.v.. Trong số đó có 2 ngôn ngữ thông dụng là **DTD**, **XML Schema**.

Đặc điểm của DTD:

- Ra đời rất sớm
- Cho phép mô tả văn bản có cấu trúc bất kỳ
- Đơn giản, dễ học và sử dụng
- Chỉ cho phép đặc tả một số “**kiểu dữ liệu đơn giản**” trong nội dung chính của tài liệu XML

Đặc điểm của XML Schema:

- Được đề xuất bởi W3C
- Chỉ áp dụng cho tài liệu XML
- Khó học và sử dụng so với DTD
- Cho phép đặc tả **chi tiết** về các “**kiểu dữ liệu**” được sử dụng trong nội dung chính của tài liệu XML

```

<?xml version="1.0" encoding="utf-8"?>
<PHAN_SO>
    <Tu_so> 4 </Tu_so>
    <Mau_so> 3 </Mau_so>
</PHAN_SO>

```

Đặc tả với DTD:

```

<!DOCTYPE PHAN_SO [
<!ELEMENT PHAN_SO (Tu_so, Mau_so) >
<!ELEMENT Tu_so #PCDATA >
    <!-- Tu_so : Số nguyên //0 -->
<!ELEMENT Mau_so #PCDATA>
    <!-- Mau_so : Số nguyên //0 -->
]

```

Đặc tả với Xml Schema:

```

<?xml version="1.0" encoding="utf-8"?>
<xsschemaid="PHAN_SO" targetNamespace="http://tempuri.org/PHAN\_SO.xsd" xmlns:xss="http://www.w3.org/2001/XMLSchema">

<xss:elementname="PHAN_SO" type="PHAN_SO"/>
<xss:complexType name="PHAN_SO">
<xss:sequence>
    <xss:elementname="Tu_so" type="SO_NGUYEN_DUONG" minOccurs="1" maxOccurs="1"/>
    <xss:elementname="Mau_so" type="SO_NGUYEN_DUONG" minOccurs="1" maxOccurs="1" />
</xss:sequence>
</xss:complexType>

<xss:simpleType name="SO_NGUYEN_DUONG">
    <xss:restrictionbase="xs:int">
        <xss:minExclusivevalue="0"/>
        <xss:restriction>
    </xss:restriction>
</xss:simpleType>
</xss:schema>

```

Trường hợp 1: với trường hợp này tài liệu đặc tả cấu trúc được sử dụng như phương tiện giao tiếp giữa các chuyên viên tin học có liên quan đến tài liệu XML tương ứng.

Có thể được lưu trữ theo bất kỳ định dạng nào thích hợp cho việc sử dụng (trình bày, xem báo cáo , v.v..).

Sử dụng đặc tả cấu trúc

Ý nghĩa của đặc tả cấu trúc: Có 2 trường hợp chính cần thiết sử dụng các tài liệu đặc tả cấu trúc:

- Trường hợp 1 : Sử dụng cho việc trao đổi thông tin **người – người**.
- Trường hợp 2 : Sử dụng cho việc trao đổi thông tin **người – hệ thống xử lý**.

Trường hợp 1: với trường hợp này tài liệu đặc tả cấu trúc được sử dụng như phương tiện giao tiếp giữa các chuyên viên tin học có liên quan đến tài liệu XML tương ứng.

Có thể được lưu trữ theo bất kỳ định dạng nào thích hợp cho việc sử dụng (trình bày, xem báo cáo , v.v..).

Ví dụ: Có thể sử dụng các tài liệu đặc tả cấu trúc (DTD/ XML Schema trên) trong:

- Hồ sơ thiết kế phần mềm hay giáo trình này (theo dạng tập tin của Microsoft Word)
- Tài liệu mô tả cách thức trao đổi thông tin giữa các chuyên viên tin cùng xây dựng các phần mềm bài tập phân số.

Có thể có một số **qui ước riêng** mang tính cục bộ trong một nhóm, có thể mở rộng các ngôn ngữ đặc tả cấu trúc hiện có để bổ sung thêm các từ vựng, cú pháp và ngữ nghĩa riêng.

Trường hợp 2: chỉ được sử dụng khi Có hệ thống xử lý (phần mềm, hàm , đối tượng thư viện) “hiểu” và thực hiện các xử lý tương ứng nào đó với tài liệu đặc tả cấu trúc.

Xử lý thông dụng nhất là **kiểm tra một tài liệu XML có theo đúng cấu trúc được mô tả trong tài liệu đặc tả cấu trúc hay không**.

Namespace

Namespace giúp cho việc truy xuất đến các thành phần (*Element*) một cách tương minh.

Namespace là **tập hợp các tên dùng để cho phép kết hợp với các thành phần và thuộc tính bên trong một tài liệu XML** nhằm giải quyết nguy cơ xung đột về tên của các phần tử khi thông tin được tổng hợp từ nhiều nguồn khác nhau.

Thông qua Namespace, trình duyệt có thể **kết hợp các file XML từ nhiều nguồn khác nhau**, có thể truy xuất đến DTD để kiểm tra cấu trúc của XML nhận được có thực sự thích hợp, từ đó xác định được tính hợp lệ của XML tương ứng.

Giải quyết xung đột:

```
<p:table length="2.5m" width="1.2m" height="0.9m">
  <p:name> Italian coffee style </p:name>
  <p:material> training oval wood
  </p:material>
</p:table>
<s:table width="100%" height="80%">
  <s:tr>
    <s:td>Orange</s:td>
    <s:td>Strawberry</s:td>
  </s:tr>
</s:table>
```

Namespace

Cú pháp khai báo namespace và thuộc tính **xmlns**:

```
<nameSpacePrefix:elementName xmlns:nameSpacePrefix = "URI">
```

...

```
</nameSpacePrefix:elementName>
```

nameSpacePrefix: phần viết tắt đại diện cho nameSpace được sử dụng như là tiền tố (prefix) cho các tag trong cùng nhóm.

xmlns: là thuộc tính được sử dụng để khai báo và chỉ ra nameSpace cần thiết sẽ áp dụng trong cấu trúc XML.

URI (Uniform Resource Identifier): chuỗi ký tự mô tả cho 1 nguồn tài nguyên nào đó duy nhất trên Internet.

```
<?xml version="1.0" encoding=" ISO-8859-1" standalone="yes"?>
<dataCombination xmlns:p="http://www.bodua.com/furniture/"
  xmlns:s="http://www.bodua.com/statistics/">
  <p:table length="2.5m" width="1.2m" height="0.9m">
    <p:name> Italian coffee style </p:name>
    <p:material> training oval wood </p:material>
  </p:table>
  <s:table width="100%" height="80%">
    <s:tr>
      <s:td>Orange</s:td>
      <s:td>Strawberry</s:td>
    </s:tr>
  </s:table>
</dataCombination>
```

DTD

Đặc tả cấu trúc tài liệu XML với DTD

Có nhiều dạng khác nhau cho phép khai báo (đặc tả) cấu trúc của tài liệu XML:
Dạng 1: Khai báo cấu trúc tài liệu XML được lưu trữ **ngay bên trong** chính tài liệu XML đó:

```
<!DOCTYPE Ten_the_goc [  
    đặc tả cấu trúc nội dung các thẻ  
    đặc tả thuộc tính các thẻ  
>]
```

Dạng 2: Khai báo cấu trúc tài liệu XML được lưu trữ **bên ngoài dưới dạng một tập tin** chứa đặc tả cấu trúc nội dung các thẻ, đặc tả thuộc tính các thẻ. Cú pháp:

```
<!DOCTYPE Ten_the_goc SYSTEM Ten_tap_tin >
```

Ví dụ :

```
<!DOCTYPE DUONG_TRON SYSTEM "DUONG_TRON.dtd" >
```

Đặc tả cấu trúc nội dung thẻ

Cú pháp chung **đặc tả cấu trúc nội dung** của một thẻ:

```
<!ELEMENT Ten_the Bieu_thuc_dac_ta_cau_truc_noi_dung >
```

Bieu_thuc có thể chỉ là một từ khoá

Bieu_thuc cũng có thể bao gồm nhiều từ khóa khác mô tả cách bố trí, sắp xếp các thành phần con bên trong thẻ

Với A, B là 2 thẻ con của thẻ X:

A, B A, B sắp xếp theo thứ tự tuần tự A đến B

A* A có thể lặp lại ít nhất 0 lần (>=0)

B+ B có thể lặp lại ít nhất 1 lần(>=1)

A? A có thể xuất hiện 0 hoặc 1 lần (0 or1)

A|B Có thể chọn sử dụng A hay B

Đặc tả cấu trúc nội dung thẻ DTD

Đặc tả cách 1:

Từ khóa ANY : Thẻ có nội dung bất kì theo định chuẩn XML. Ví dụ :

```
<!ELEMENT X ANY >
```

- X có thể chứa nội dung bất kỳ, khai báo này chỉ để mô tả sự tồn tại của bên trong X một hoặc nhiều thẻ khác.

Từ khóa EMPTY : Thẻ không có nội dung. Ví dụ :

```
<!ELEMENT PHAN_SO EMPTY >
```

- PHAN_SO không thể có nội dung mà chỉ có thể có các thuộc tính.

Từ khóa #PCDATA : Thẻ với nội dung là chuỗi văn bản. Ví dụ :

```
<!ELEMENT Ho_ten (#PCDATA) >
```

- Ho_ten có nội dung là chuỗi và không thể chứa các thẻ khác

Với DTD muốn mô tả chi tiết hơn, dùng thẻ ghi chú. Ví dụ :

```
<!ELEMENT He_so (#PCDATA) >
```

```
<!-- He_so : A_Float -->
```

Đặc tả cách 2:

Dạng tuần tự: Các thẻ con chỉ có thể xuất hiện 1 lần duy nhất và phải theo đúng thứ tự xuất hiện trong biểu thức

Cú pháp :

```
<!ELEMENT Ten_the (Ten_the_1, Ten_the_2, ....) >
```

Ý nghĩa : The_1, The_2, ..., The_k phải xuất hiện một lần duy nhất theo đúng thứ tự trên. Ví dụ:

```
<!ELEMENT DON_THUC(He_so, So_mu) >
```

Thẻ DON_THUC phải bao hàm bên trong 2 thẻ con He_so,So_mu theo đúng thứ tự trên

Ghi chú: Các thẻ bên trong có thẻ có tên trùng nhau. Ví dụ :

```
<!ELEMENT TAM_GIAC (DIEM, DIEM, DIEM) >
```

Có thể sử dụng từ khóa #PCDATA trong biểu thức tuần tự (và các loại biểu thức khác). Ví dụ :

```
<!ELEMENT X (#PCDATA, A, #PCDATA)>
```

- Thẻ X phải bao gồm 3 thành phần :
- Thành phần thứ 1 là chuỗi văn bản, thành phần thứ 2 là thẻ có tên A, thành phần thứ 3 là chuỗi văn bản

Dạng tùy chọn: Thẻ con có thể được sử dụng hay không sử dụng. Cú pháp (dạng đơn giản) :

```
<!ELEMENT Ten_the ( Ten_the_con ? ) >
```

- Thẻ đang xét có thẻ chứa 1 hay 0 lần xuất hiện của thẻ có tên là Ten_the_con

Ví dụ :

```
<!ELEMENT DON_THUC (Ten?) >
```

Thẻ DON_THUC có thể chứa hay không thẻ Ten

Có thể kết hợp với biểu thức tuần tự:

```
<!ELEMENT X (A,B?,C) >
```

Có thể cho phép tùy chọn một tập hợp các thẻ:

```
<!ELEMENT X (A,B,C)? >
```

- X có thể chứa bên trong các thẻ A,B,C (theo thứ tự trên) hay cũng có thẻ không chứa bất kỳ thẻ nào

Dạng chọn : Bắt buộc chọn một thẻ con để sử dụng trong tập hợp thẻ cho trước. Cú pháp:

```
<!ELEMENT: Ten_the(Ten_the_1|Ten_the_2|..|Ten_the_k) >
```

Có thể kết hợp với biểu thức tuần tự:

```
<!ELEMENT X (A,B|C,D) >
```

- Thành phần đầu tiên của thẻ X là thẻ A, kế đến là thẻ B hay thẻ C và thành phần cuối cùng phải là D

Có thể cho phép chọn một tập hợp các thẻ

```
<!ELEMENT X ( (A,B) | (C,D) ) >
```

- X có thể bao hàm bên trong cặp thẻ A,B (theo thứ tự trên) hay cặp thẻ C,D (theo thứ tự trên)

Dạng lặp:

Dạng lặp ít nhất 0 lần : Các thẻ con có thể lặp lại nhiều lần hay có thẻ không có lần nào. Cú pháp :

```
<!ELEMENT Ten_the (Ten_the_con*) >
```

Ý nghĩa: Thẻ đang xét có thể bao hàm bên trong nhiều thẻ có tên là Ten_the_con hay cũng có thể là thẻ rỗng (không có nội dung)

Ví dụ :

```
<!ELEMENT LOP (HOC_SINH*) >
```

- Thẻ LOP có thể chứa nhiều thẻ HOC_SINH hay không có thẻ HOC_SINH nào

Có thể mô tả lặp đồng thời nhiều thẻ con

```
<!ELEMENT X (A,B,C)* >
```

- Các thẻ A,B,C theo thứ tự trên có thể lặp lại ít nhất 0 lần trong thẻ X

Có thể kết hợp với biểu thức tuần tự. Ví dụ :

```
<!ELEMENT X (A,B*,C) >
```

- Thẻ X có thành phần đầu tiên là thẻ A, kế đến có thẻ có nhiều hay 0 lần lặp của thẻ B và cuối cùng là thẻ C

Có thể kết hợp với biểu thức tùy chọn. Ví dụ :

```
<!ELEMENT X (A,B*,C?,D) >
```

Dạng lặp ít nhất 1 lần : Các thẻ con có thể lặp lại nhiều lần và ít nhất là một lần.

Cú pháp :

```
<!ELEMENT Ten_the (Ten_the_con+)>
```

Ý nghĩa : Thẻ đang xét có thể bao hàm bên trong ít nhất một thẻ có tên là Ten_the_con. Ví dụ :

```
<!ELEMENT DA_THUC (DON_THUC+)>
```

- Thẻ DATHUC phải bao hàm bên trong ít nhất một thẻ DON_THUC

Có thẻ mô tả lặp đồng thời nhiều thẻ con

```
<!ELEMENT CT_HOA_DON (Mat_hang,So_luong,Don_gia) +>
```

- Các thẻ CT_HOA_DON phải bao hàm ít nhất 1 lần 3 thẻ Mat_hang,So_luong,Don_gia

Có thẻ kết hợp với biểu thức tuần tự. Ví dụ

```
<!ELEMENT DA_GIAC  
(DIEM,DIEM,DIEM+)>
```

- Các thẻ DA_GIAC phải bao hàm ít nhất 3 thẻ DIEM

Cú pháp khai báo **đặc tả thuộc tính** chung:

```
<!ATTLIST Ten_the
```

```
Ten_thuoc_tinh_1 Kieu_1 Tham_so_1  
Ten_thuoc_tinh_2 Kieu_2 Tham_so_2
```

...

```
Ten_thuoc_tinh_k Kieu_k Tham_so_k
```

```
>
```

Ý nghĩa :

Ten_the : tên thẻ cần khai báo các thuộc tính

Ten_thuoc_tinh_1, Ten_thuoc_tinh_2, ... Ten_thuoc_tinh_k : Tên các thuộc tính của thẻ

Kieu_1, Kieu_2, ..., Kieu_k : Mô tả tập hợp các giá trị mà thuộc tính có thể nhận

Tham_so_1, Tham_so_2, ..., Tham_so_k : Mô tả một số tính chất trên thuộc tính tương ứng

Ví dụ: Đặc tả cấu trúc tài liệu XML biểu diễn thông tin về biểu thức phân số:

$$P = 4/5 + 6/7 * 1/3 - 10/3 + 11/2 * 2/3$$

```
<?xml version="1.0" encoding="utf-8"?>  
<!DOCTYPE BIEU_THUC [  
  <!ELEMENT BIEU_THUC (PHAN_SO | TICH_SO)+>  
    <!ATTLIST BIEU_THUC Ten CDATA #IMPLIED  
      <!-- Ten : A_String -->  
    >  
  <!ELEMENT PHAN_SO EMPTY>  
  <!ATTLIST PHAN_SO  
    Tu_so CDATA #REQUIRED  
      <!-- Tu_so : A_Int -->  
    Mau_so CDATA #REQUIRED  
      <!-- Mau_so : A_Int // >0 -->  
    >  
  <!ELEMENT TICH_SO (PHAN_SO)+>  
]>
```

Kiểu thuộc tính : Mô tả **tập hợp các giá trị** của thuộc tính. Một số cách thông dụng mô tả:

Cách 1 : Dùng từ khoá **CDATA**

Cú pháp :

```
<!ATTLIST Ten_the
```

...

```
Ten_thuoc_tinh CDATA
```

...

```
>
```

Tập hợp các giá trị của thuộc tính với khai báo CDATA chính là **tập hợp các chuỗi**

Ví dụ : Đặc tả cấu trúc tài liệu XML biểu diễn phương trình đường thẳng trong mặt phẳng

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE DUONG_THANG [
<!ELEMENT DUONG_THANG EMPTY>
<!--phương trình ax + by+c =0 -->
```

<ATTLIST DUONG_THANG

```
Ten CDATA #IMPLIED
    <!-- Ten : A_String -->
a CDATA #REQUIRED
    <!-- a : A_Float-->
b CDATA #REQUIRED
    <!-- b : A_Float-->
c CDATA #REQUIRED
    <!-- c : A_Float-->
```

>

```
    <!--a,b không đồng thời là 0 -->
```

đ>

Cách 2: dùng **biểu thức liệt kê**. Cú pháp :

```
<!ATTLIST Ten_the
...
Ten_thuoc_tinh ( Gia_tri_1,Gia_tri_2,...,Gia_tri_k) ...
```

>

Ý nghĩa : Tập hợp các giá trị có thể có của thuộc tính đang xét chính là tập hợp các giá trị được liệt kê

Gia_tri_1,Gia_tri_2, ...,Gia_tri_k: Các giá trị này là các chuỗi ký tự

Ví dụ: Đặc tả cấu trúc tài liệu XML biểu diễn thông tin về phiếu điểm của một học sinh:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE PHIEU_DIEM [
<!ELEMENT PHIEU_DIEM (HOC_SINH, DIEM_SO+ ) >
```

<!ELEMENT HOC_SINH EMPTY >

```
<ATTLIST HOC_SINH
Ho_ten CDATA #REQUIRED
    <!-- Ho_ten : A_String -->
Ngay_sinh CDATA #REQUIRED
    <!--Ngay_sinh : A_Date -->
Xep_loai("Giỏi", "Khá", Trung bình", Yếu")#IMPLIED
>
```

<!ELEMENT DIEM_SO EMPTY >

```
<ATTLIST DIEM
Ten_mon CDATA #REQUIRED
    <!-- Ten_mon : A_String -->
Gia_tri CDATA #REQUIRED
    <!-- Gia_tri : A_Float // từ 0 đến 10 -->
```

>

Tham số của thuộc tính: mô tả tính chất của thuộc tính, một số cách mô tả:

Cách 1: Dùng từ khóa **#REQUIRED**. Cú pháp :

```
<!ATTLIST Ten_the
...
Ten_thuoc_tinh Kieu #REQUIRED
...
>
```

Ý nghĩa : Thuộc tính đang xét là thuộc tính **bắt buộc** phải có. Đây là cách sử dụng phổ biến nhất

Ví dụ : Đặc tả cấu trúc tài liệu XML biểu diễn thông tin về các đơn thức với tên bắt buộc phải có

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE DON_THUC [
<!ELEMENT DON_THUC (He_so, So_mu ) >
<!ATTLIST DON_THUC
    Ten   CDATA #REQUIRED
        <!-- Ten : A_String -->
    Bien_so  CDATA #REQUIRED
        <!-- Bien_so: A_String      -->
>
<!ELEMENT He_so #PCDATA >
    <!-- He_so : A_Float       -->
<!ELEMENT So_mu #PCDATA >
    <!-- So_mu : A_Int // >=0 -->
]>
```

Cách 2 : Dùng từ khóa **#IMPLIED**. Cú pháp :

```
<!ATTLIST Ten_the
...
Ten_thuoc_tinh Kieu #IMPLIED
...
>
```

Ý nghĩa : Thuộc tính đang xét là tùy chọn và không bắt buộc phải có.

Ví dụ : Đặc tả cấu trúc tài liệu XML biểu diễn thông tin về tam thức $P(x) = 2x^2 - 4x + 6$.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE TAM_THUC [
<!ELEMENT TAM_THUC (DON_THUC,DON_THUC,DON_THUC) >
<!ATTLIST TAM_THUC
    Ten   CDATA #IMPLIED
        <!-- Ten : A_String
    Bien_so  CDATA // sinh sẵn là x -->
>
<!ELEMENT DON_THUC EMPTY >
<!ATTLIST DON_THUC
    He_so  CDATA #REQUIRED
        <!-- He_so : A_Float // Khác 0 nếu So_mu=2      -->
    So_mu (0,1,2) #REQUIRED
        <!-- So_mu : A_Int // =0,1,2 và khác nhau -->
>
]>
```

Cách 3: Dùng từ khóa **#FIXED**. Cú pháp :

```
<!ATTLIST Ten_the
...
Ten_thuoc_tinh Kieu #FIXED Gia_tri
...
>
```

Ý nghĩa : Thuộc tính đang xét phải có giá trị cố định là **Gia_tri**. Trường hợp này ít được sử dụng

Ví dụ :Đặc tả cấu trúc tài liệu XML biểu diễn thông tin về các đơn thức chỉ với biến số x:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE DON_THUC [
<!ELEMENT DON_THUC (He_so, So_mu ) >
<!ATTLIST DON_THUC
    Ten   CDATA #REQUIRED
        <!-- Ten : A_String      -->
    Bien_so CDATA      #FIXED "x"
        <!-- Bien_so: A_String -->
>

<!ELEMENT He_so (#PCDATA) >
    <!-- He_so : A_Float     -->

<!ELEMENT So_mu (#PCDATA) >
    <!-- So_mu : A_Int // >=0 -->
]>
```

XML Schema

XML Schema: khai báo XML Schema là tạo lập tài liệu XML mà nội dung chính là các thẻ đánh dấu, các thẻ này mô tả cho cấu trúc các thẻ của một tài liệu XML khác. Cú pháp:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    Đặc tả các thẻ
    Đặc tả các kiểu
</xs:schema>
```

Thông tin về một thẻ được mô tả tập trung qua một phương cách duy nhất là **kiểu**.

Mỗi thẻ sẽ có tương ứng một kiểu, đặc tả kiểu mô tả kiểu của thẻ cùng với một số thông tin khác chính là **cách sắp xếp các thành phần bên trong** của thẻ và **hệ thống các thuộc tính** của thẻ.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="DA_THUC" type="K_DA_THUC"/>

    <xs:complexType name="K_DA_THUC">
        <xs:sequence>
            <xs:element name="DON_THUC" type="K_DON_THUC" minOccurs="1"/>
        </xs:sequence>
        <xs:attribute name="Ten" type="xs:string" />
        <xs:attribute name="Bien_so" type="xs:string"/>
    </xs:complexType>
```

Tài liệu XML có thẻ gốc là **DA_THUC** thẻ này có kiểu là **kiểu phức hợp** tên là **K_DA_THUC** (có thể dùng cùng tên là DA_THUC).

Kiểu phức hợp K_DA_THUC bao gồm bên trong:

- Thẻ **DON_THUC** có kiểu là **kiểu phức hợp** và thẻ DON_THUC phải xuất hiện ít nhất 1 lần
- 2 thuộc tính :
Ten với kiểu là **kiểu cơ sở dạng chuỗi**
Bien_so với kiểu là **kiểu cơ sở dạng chuỗi**

```

<xs:complexType name="K_DON_THUC">
<xs:attribute name="He_so" type="xs:float"/>
<xs:attribute name="So_mu" type="SO_TU_NHIEN"/>
</xs:complexType>
<xs:simpleType name="SO_TU_NHIEN">
    <xs:restriction base="xs:int">
        <xs:minInclusive value="0"/>
    </xs:restriction>
</xs:simpleType>
</xs:schema>

```

Kiểu phức hợp K_DON_THUC không có nội dung và chỉ bao gồm các thuộc tính:

- He_so có kiểu là **kiểu cơ sở loại số thực**
- So_mu có kiểu là **kiểu đơn giản** với tên SO_TU_NHIEN

Kiểu đơn giản SO_TU_NHIEN chính là kiểu cơ sở số nguyên với hạn chế: giá trị phải lớn hơn hay bằng 0

Kiểu định nghĩa sẵn – thư viện

Ý nghĩa sử dụng :

- Được sử dụng để mô tả trực tiếp kiểu của các thuộc tính hay của thẻ thỏa 2 điều kiện :
 - Điều kiện 1 : thẻ không có thuộc tính
 - Điều kiện 2 : thẻ không chứa thẻ khác (nội dung là chuỗi văn bản) và có miền giá trị (tập hợp giá trị có thể có) thích hợp với kiểu

Ten kieu co so	Ý nghĩa
string	Chuỗi ký tự
int, integer	Số nguyên
float	Số thực chính xác đơn
double	Số thực chính xác kép
boolean	Giá trị logic
date	ngày
month	Tháng
ID	Chuỗi định danh
binary	Dữ liệu nhị phân

Cú pháp :

- Khi dùng với thẻ:

```
<xs:element name="Ten_the" type="Ten_kieu_co_so" ... />
```

- Khi dùng với thuộc tính:

```
<xs:attribute name="Ten_thuoc_tinh" type="Ten_kieu_co_so" .. />
```

Ví dụ:

```

<xs:element name="Ho_ten" type="xs:string" />
<xs:element name="Ngay_sinh" type="xs:date" />
<xs:attribute name="He_so" type="xs:float"/>
<xs:attribute name="f" type="xs:boolean"/>

```

XML Schema có 3 loại kiểu chính :

- Loại 1 : Kiểu định nghĩa sẵn (BuiltinType)
- Loại 2 : Kiểu đơn giản (simpleType)
- Loại 3 : Kiểu phức hợp (complexType).

XML Schema kiểu đơn giản

Kiểu đơn giản (**simpleType**): Là các kiểu do người dùng định nghĩa dựa trên các kiểu cơ sở có sẵn.

Ý nghĩa sử dụng: để mô tả trực tiếp kiểu của các thuộc tính hay các thẻ thỏa mãn:

- Điều kiện 1 : Không có thuộc tính
- Điều kiện 2 : Không chứa thẻ khác (nội dung là chuỗi văn bản) và có miền giá trị (tập hợp giá trị có thể có) là tập con của miền giá trị một kiểu cơ sở nào đó

Cú pháp:

```
<xs:simpleType name="Ten_kieu">
<xs:restriction base="Ten_kieu_co_so">
    Giới hạn ( ràng buộc ) trên miền giá trị
</xs:restriction>
</xs:simpleType>
```

Khai báo cận dưới: Sử dụng từ khoá **minInclusive** (cận dưới cho phép sử dụng biên), **minExclusive** (cận dưới không cho phép sử dụng biên)

Khai báo cận trên: Sử dụng từ khoá **maxInclusive** (cận trên cho phép sử dụng biên), **maxExclusive** (cận trên không cho phép sử dụng biên)

```
<xs:simpleType name="KY_SO">
<xs:restriction base="xs:int">
    <xs:minInclusive value="0" />
    <xs:maxInclusive value="9" />
</xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="DIEM_SO">
<xs:restriction base="xs:float">
    <xs:minInclusive value="0" />
    <xs:maxInclusive value="10" />
</xs:restriction>
</xs:simpleType>
```

Giới hạn loại **liệt kê** trên kiểu cơ sở: Cho phép xác định miền giá trị của kiểu đơn giản bằng cách liệt kê các giá trị. Cú pháp:

```
<xs:simpleType name="Ten_kieu">
    <xs:restriction base="Ten_kieu_co_so_loai_so">
        <xs:enumeration value="Gia_tri_1" />
        <xs:enumeration value="Gia_tri_2" />
        ...
        <xs:enumeration value="Gia_tri_k" />
    </xs:restriction>
</xs:simpleType>
```

```

<xs:simpleType name="LOAI_HOC_LUC" >
  <xs:restriction base="xs:string">
    <xs:enumeration value="Giỏi" />
    <xs:enumeration value="Khá" />
    <xs:enumeration value="Trung bình" />
    <xs:enumeration value="Yếu" />
  </xs:restriction>
</xs:simpleType>

```

XML Schema kiểu phức hợp

Kiểu phức hợp complexType: Là các kiểu do người dùng tự định nghĩa cho phép mô tả nội dung và các thuộc tính của các thẻ được khai báo thuộc về kiểu đang xét.

Ý nghĩa sử dụng :

- Được sử dụng để mô tả kiểu của các thẻ thỏa một trong 2 điều kiện :
 - Điều kiện 1 : Có thuộc tính
 - Điều kiện 2 : Có chứa thẻ khác

Cú pháp chung:

```

<xs:complexType name="Ten_kieu"> Dac_ta_cau_truc_noi_dung
  Dac_ta_thuoc_tinh
</xs:complexType>

```

Dac_ta_cau_truc_noi_dung : Mô tả cách thức tổ chức, sắp xếp các thẻ con bên trong. Tương tự như DTD, XML Shema cho phép nhiều dạng tổ chức sắp xếp (tuần tự, chọn, lặp) các thẻ con.Ngoài ra cho phép khai báo chi tiết hơn về số lần lặp của một thành phần.

Dac_ta_thuoc_tinh: Mô tả hệ thống các thuộc tính của thẻ. Việc mô tả các thuộc tính trong XML Shema cũng tương tự như mô tả thuộc tính trong DTD nhưng cho phép định nghĩa và sử dụng các kiểu đơn giản để mô tả chi tiết về miền giá trị của một thuộc tính.

Đặc tả cấu trúc nội dung dạng **tuần tự**:

Dạng tuần tự : Sử dụng thẻ/từ khóa **sequence**. Cú pháp :

```

<xs:complexType name="Ten_kieu">
  <xs:sequence> Thanh_phan_1
    Thanh_phan_2
    Thanh_phan_k
  </xs:sequence>
  ....
</xs:complexType>

```

Ví dụ:

```

<xs:complexType name="DIEM">
  <xs:sequence>
    <xs:element name="x" type="xs:float" />
    <xs:element name="y" type="xs:float" />
  </xs:sequence>
</xs:complexType>

```

Đặc tả cấu trúc nội dung dạng **tùy chọn**:

Dạng tùy chọn : Sử dụng thẻ/từ khóa **choice**

Cú pháp :

```

<xs:complexType name="Ten_kieu">
  <xs:choice> Thanh_phan_1
    Thanh_phan_2
    Thanh_phan_k
  </xs:choice>
  ....
</xs:complexType>

```

Ví dụ :

```
<xs:complexType name="X">
  <xs:choice>
    <xs:element name="A" type="A" />
    <xs:element name="B" type="xs:string" />
  </xs:choice>
</xs:complexType>
```

Đặc tả cấu trúc nội dung dạng **lặp**:

Dạng lặp : Sử dụng thuộc tính/từ khóa **minOccurs** , **maxOccurs**. Cú pháp:

```
<xs:complexType name="Ten_kieu">
  <xs:sequence>
    ...
    <xs:element name="Ten_the_con" type="Kieu_the_con"
      minOccurs="So_lan_lap_toi_thieu" maxOccurs="So_lan_lap_toi_da" />
    ...
  </xs:sequence>
  ...
</xs:complexType>
```

Ví dụ:

```
<xs:complexType name="DA_THUC">
  <xs:sequence>
    <xs:element name="DON_THUC"
      type="DON_THUC" minOccurs="1" />
  </xs:sequence>
  <!-- Mô tả các thuộc tính -->
  ...
</xs:complexType>
```

```
<xs:complexType name="DA_GIAC">
  <xs:sequence>
    <xs:element name="DIEM" type="DIEM"
      minOccurs="3"
      maxOccurs="3" />
  </xs:sequence>
  <!-- Mô tả các thuộc tính -->
  ...
</xs:complexType>
```

Đặc tả **thuộc tính**: cho phép mô tả hệ thống các thuộc tính của một thẻ
Cú pháp :

```
<xs:complexType name="Ten_kieu">
  Đặc tả cấu trúc nội dung
  ...
  <xs:attribute name="Ten_thuoc_tinh" type="Kieu_thuoc_tinh"
    Tinh_chat_thuoc_tinh />
  ...
</xs:complexType>
```

Tinh_chat_thuoc_tinh : Mô tả một số tính chất của thuộc tính, mỗi tính chất
tương ứng với một từ khóa riêng.

```
<xs:attribute name="Ten_thuoc_tinh" type="Kieu_thuoc_tinh"
  Tu_khoa_1="Gia_tri_1"
  Tu_khoa_2="Gia_tri_2"
  Tu_khoa_k="Gia_tri_k" />
```

Một số tính chất thông dụng:

- Giá trị định sẵn : từ khóa **default**
- Giá trị cố định: từ khóa **fixed**
- Tùy chọn (có hay không có) sử dụng : từ khóa **use**

Đặc tả thẻ: các thông tin cần mô tả khi đặc tả một thẻ trong XML bao gồm:

- Tên thẻ
- Kiểu của thẻ
- Một số tính chất khác của thẻ

Cú pháp khai báo:

```
<xs:element name="Ten_the" type="Ten_kieu" Thuoc_tinh_khac />
```

Tên của kiểu: mô tả thông tin về thẻ, tên kiểu và tên thẻ được đặt trùng nhau.

Thuộc tính của thẻ: mô tả các tính chất của thẻ, thông dụng nhất là **minOccurs**, **maxOccurs**.

Khi đặc tả các thẻ vấn đề quan trọng nhất là **xác định loại kiểu** sẽ dùng trong thẻ.

Phân loại thẻ: 2 nhóm chính:

- Nhóm 1 : Nhóm các thẻ có thuộc tính
- Nhóm 2 : Nhóm các thẻ không có thuộc tính

Với các thẻ **có thuộc tính**, nhất thiết phải sử dụng **kiểu phức hợp**

- => Khai báo kiểu phức hợp Y (có thẻ dùng tên thẻ đang xét)
- => Sử dụng Y là kiểu của thẻ đang xét

Các thẻ không có thuộc tính bao gồm 2 nhóm:

- Nhóm 2.1 : Nhóm các thẻ không có thuộc tính và có chứa các thẻ con bên trong => phải sử dụng **kiểu phức hợp**
- Nhóm 2.2 : Nhóm các thẻ không có thuộc tính và không chứa các thẻ con bên trong (nội dung là chuỗi văn bản)
Có thể chọn sử dụng **kiểu cơ sở** hay **kiểu đơn giản** phụ thuộc vào miền giá trị của chuỗi văn bản bên trong thẻ

Thuật giải đặc tả thẻ: Xét loại kiểu của A:

A là **kiểu phức hợp**, đặc tả kiểu phức hợp A bao gồm:

Đặc tả **hệ thống các thẻ con** của thẻ gốc X

- Đặc tả thẻ X1 với thông tin về kiểu (giả sử là A1)
- Đặc tả thẻ X2 với thông tin về kiểu (giả sử là A2)
- ...
- Đặc tả thẻ XK với thông tin về kiểu (giả sử là Ak)

Đặc tả **hệ thống các thuộc tính** của thẻ gốc X

- Đặc tả thuộc tính T1 với thông tin về kiểu (giả sử là B1)
- Đặc tả thuộc tính T2 với thông tin về kiểu (giả sử là B2)
- Đặc tả thuộc tính Tk với thông tin về kiểu (giả sử là Bk)

A là **kiểu đơn giản**: Đặc tả kiểu đơn giản A bao gồm

- Đặc tả kiểu cơ sở của A
- Đặc tả các hạn chế trên kiểu cơ sở của A

A là **kiểu cơ sở**:

Không cần Đặc tả thêm

Xét loại kiểu của A1

Xét loại kiểu của A2

...

Xét loại kiểu của Ak

Xét loại kiểu của B1

Xét loại kiểu của B2

...

Xét loại kiểu của Bk

Xét loại kiểu của T1

Xét loại kiểu của T2

...

Xét loại kiểu của Tk

.....

Xét loại kiểu của các kiểu phát sinh thêm khi đặc tả các kiểu phía trên

.....