**Three Phases of Event Dispatch**

|  |
| --- |
| •Capturing – the event travels downward from the document object to the target element  •Target – the event triggers on the target element  •Bubbling – the event travels upward from the target element to the document object |

**Two ways of event:**

|  |
| --- |
| 1:document.onreadystatechange = function () {  if (document.readyState == "complete") { initApplication(); }  }  2: false 代表是bubble, true代表capture,默认是false  window.addEventListener( "load", startTimer, false ); |

**Json and XML:**

|  |
| --- |
| **1:Json: values could be number,string, Boolean,array,object, null**  {"employees":[     {"firstName":"John", "lastName":"Doe"},     {"firstName":"Anna", "lastName":"Smith"}, ]}  **2: parse the Json string to javascript object**  var text = '{ "employees" : [' + '{ "firstName":"John" , "lastName":"Doe" },' + '{ "firstName":"Anna" , "lastName":"Smith" },' + '{ "firstName":"Peter" , "lastName":"Jones" } ]}';  var obj = JSON.parse(text);  **3: json to javascript array**  var nums = '["200","400","600","800"]';  var anarray = JSON.parse(nums);  var sum = 0;  for (i = 0; i < anarray.length; i++) {  sum += parseInt(anarray[i]);  }  document.getElementById("result").innerHTML = sum; |

**Ajax: Asynchronous Javascript and XML**

|  |
| --- |
| **1:why?**  Enables web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.  **2: how it works?**    **3: onreadystatechange:** When status == 200, and state =4, we have obtained the respose from our initial request.    **4: Example on the example**  1.Get Request from Browser  2.Server Sends back HTML  3.HTML page is loaded and rendered in your browser  4.Onclick event of Change Content Button set to call the JavaScript function “loadXMLdoc”  5.Create an XMLHttpRequest object (stored in xmlhttp variable)  6.Call the open method on xmlhttp variable with the request type (GET, the url of the file you are getting, and whether the request is asynchronous or not (true is asynchronous, false is synchronous)  This time, the file is an XML file, not a text file as before  7.Send the GET request (note, this is an HTTP request to the Server)  8.Assign a callback function that does the work with whatever is returned to the xmlhttp.onreadystatechange attribute – in this case we call the responseXML method on the xmlhttp object to get the XML document  9.Then get an array objects in the XML DOM associated with the TAG artist using the getElementsByTagName method on the XML document object  10.Then we traverse the array, build an HTML string of artist names  11.The work for this example is to update the innerHTML of the div element with that string |

**XML: extensible markup language**

|  |
| --- |
| **1: XML was designed to transport and store data,**  <employees>     <employee>         <firstName>John</firstName> <lastName>Doe</lastName>     </employee>     <employee>         <firstName>Anna</firstName> <lastName>Smith</lastName>     </employee> </employees>  **2: XML vs HTML**  •XML and HTML were designed with different goals:  –XML was designed to transport and store data, with focus on what the data is  –HTML was designed to display data, with focus on how the data looks  •HTML is about displaying information, while XML is about carrying information.  **3:XML Basic Concepts**:  •XML is a meta language for defining any desired markup language for an application domain.  **•Every document must have one and only one root element.**  •–XML has no tags of its own, you define them!!!  •An XML-based markup language can be used to store and communicate information.  •Using XML, new tags and structures can be defined to describe the structure of the information to be stored and communicated.  •Newly created tags must adhere to the rules of XML specification.  •XML is flexible - The structures used for information organization can be extended, and structures developed by different organizations can be combined.  •Elements are the basic building blocks of an XML document  •An Element contains a tag, and possibly some attributes. Every tag must have a closing tag.  •An Element may contain some child elements  •You invent your own tags in XML, there are no predefined tags  **4: XML prolog**  <?xml version=”1.0” encoding=”UTF-8”?>  **5: XML namespace**      **5: XSLT: extensible stylesheet language(XSL) and XSL transpormations** |

**AJAX with XML:**

|  |
| --- |
| function loadXMLDoc()  {  var xmlhttp;  var txt,x,i;  if (window.XMLHttpRequest)  {// code for IE7+, Firefox, Chrome, Opera, Safari  xmlhttp=new XMLHttpRequest();  }  xmlhttp.onreadystatechange=function()  {  if (xmlhttp.readyState==4 && xmlhttp.status==200)  {  **xmlDoc=xmlhttp.responseXML;**  txt="";  x=xmlDoc.getElementsByTagName("ARTIST");  for (i=0;i<x.length;i++)  {  txt=txt + x[i].childNodes[0].nodeValue + "<br>";  }  document.getElementById("myDiv").innerHTML=txt;  }  } // definition of function assigned to onreadystatechange    xmlhttp.open("GET","cd\_catalog.xml",true);  xmlhttp.send();  } |

**PHP**

|  |
| --- |
| **1: what php do:**  •Generate dynamic page content  •Open, read, write, delete, and close files on the server  •Collect form data  •Send and receive cookies  •Add, delete, modify data in your database  •Restrict user access to some pages on your website  •Encrypt (and Decrypt) data  **2: why php?**  1:PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.) 2:PHP is compatible with almost all servers used today (Apache, IIS, etc.) 3:PHP supports a wide range of databases – and for this class MySQL (also free), so that is a good thing 4:PHP is free. |

**More PHP:**

|  |
| --- |
| **1:debug:**  •ini\_set('display\_errors', 'On'); error\_reporting(E\_ALL);  •var\_dump – dumps out the values of your variables  •print\_r – nicely formatted version of var\_dump  **2: variable names**   Variable names are case-sensitive.   Variable names can contain letters, numbers, and underscores.   Variable names can’t contain special characters.   Variable names can’t begin with a digit or two underscores.   Variable names can’t use names that are reserved by PHP such as the variable named $this that’s reserved for use with objects.  **3: define constant**  define(‘MAX\_QTY’,100);  define(‘PI’,3,1415);  define(‘MALE’,’m’);  **4: when use HTTP POST method**   When the request is for a page that writes data to a database server.   When executing the request multiple times may cause problems.   When you don’t want to include the parameters in the URL for security reasons.   When you don’t want users to be able to include parameters when they bookmark a page.   When you need to transfer more than 4 KB of data.  **5: PHP strings**  **Use double quotes for variable substitution**  $name = "Name: $first\_name"; // Name: Bob  $name = "$first\_name $last\_name"; // Bob Roberts  **Mix single and double quotes for special purposes**  $last\_name = "O'Brien"; // O'Brien  $line = 'She said, "Hi."'; // She said, "Hi."  **How to join a number to a string**  $price = 19.99;  $price\_string = 'Price: ' . $price; // Price: 19.99  **5: PHP number format**  $nf = number\_format(12345); // 12,345  $nf = number\_format(12345, 2); // 12,345.00  $nf = number\_format(12345.674, 2); // 12,345.67  $nf = number\_format(12345.675, 2); // 12,345.68  **6: Check the Variables**  isset($name) // TRUE if $name has been set // and is not NULL  empty($name) // TRUE if $name is empty  is\_numeric($price) // TRUE if $price is a number |

**PHP parse XML:**

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <carlot>  <car>  <make>Chevrolet</make>  <model>Camaro</model>  <year>2014</year>  <price>29995.95</price>  </car>  <car>  <make>Ford</make>  <model>Mustang</model>  <year>2014</year>  <price>28500.99</price>  </car>  </carlot>  **php:**  <?php $xml=simplexml\_load\_file(“cars.xml”) or die("Error: Cannot create object");  foreach($xml->children() as $cars) {  echo $cars->make . ", ";  echo $cars->model . ", ";  echo $cars->year . ", ";  echo $cars->price . "<br>"; } ?>  **parse XML with error handling**:  $myXMLData = "<?xml version='1.0' encoding='UTF-8'?> <document> <name>Jim Smith</nam> <address>jsmith1@lucent.com</addr> </document>";  $xml = simplexml\_load\_string($myXMLData);  if ($xml === false) {  echo "Failed loading XML: ";  foreach(libxml\_get\_errors() as $error) {  echo "<br>", $error->message; } }  else { print\_r($xml); } |

**PHP regular expression**

|  |
| --- |
| **Simple example:**  $pattern = '/Harris/';  $author = 'Ray Harris';  $author\_match = preg\_match($pattern, $author);  if ($author\_match === false) { echo 'Error testing author name.'; }  else if ($author\_match === 0) { echo 'Author name does not contain Harris.'; }  else { echo 'Author name contains Harris.'; }  **Special characters:**      $string = 'The product code is MBT-3461.';  preg\_match('/MB./', $string) // Matches MBT and returns 1  preg\_match('/MB\d/', $string) // Matches nothing and returns 0 preg\_match('/MBT-\d/', $string) // Matches MBT-3 and returns 1  **character class:**  $string = 'The product code is MBT-3461.';  preg\_match('/MB[TF]/', $string) // Matches MBT and returns 1  preg\_match('/[.]/', $string) // Matches . and returns 1  preg\_match('/[13579]/', $string) // Matches 3 and returns 1  **metacharacters:**  $string = 'The product code is MBT-3461.';  preg\_match('/MB[^TF]/', $string) // Matches nothing and returns 0 preg\_match('/MBT[^^]/', $string) // Matches MBT- and returns 1 preg\_match('/MBT-[1-5]/', $string) // Matches MBT-3 and returns 1 preg\_match('/MBT[\_\*-]/', $string) // Matches MBT- and returns 1  **bracket expression:**  $string = 'The product code is MBT-3461.';  preg\_match('/MBT[[:punct:]]/', $string) // Matches MBT- and returns 1 preg\_match('/MBT[[:digit:]]/', $string) // Matches nothing and returns 0 preg\_match('/MB[[:upper:]]/', $string) // Matches MBT and returns 1  **Patterns for string positions:**    $author = 'Ray Harris'; preg\_match('/^Ray/', $author) // Returns 1 preg\_match('/Harris$/', $author) // Returns 1  preg\_match('/^Harris/', $author) // Returns 0  $editor = 'Anne Boehm';  preg\_match('/Ann/', $editor) // Returns 1  preg\_match('/Ann\b/', $editor) // Returns 0  **matching subpatterns:**  $name = 'Rob Robertson';  preg\_match('/^(Rob)|(Bob)\b/', $name) // Returns 1  preg\_match('/^(\w\w\w) \1/', $name) // Returns 1  **matching repeating patterns:**  $phone = '559-555-6627';  preg\_match('/^\d{3}-\d{3}-\d{4}$/', $phone) // Returns 1  $fax = '(559) 555-6635';  preg\_match('/^\(\d{3}\) ?\d{3}-\d{4}$/', $fax) // Returns 1  $phone\_pattern = '/^(\d{3}-)|(\(\d{3}\) ?)\d{3}-\d{4}$/';  preg\_match($phone\_pattern, $phone) // Returns 1  preg\_match($phone\_pattern, $fax) // Returns 1  **testing a phone number:**  $phone = '559-555-6624'; $phone\_pattern = '/^[[:digit:]]{3}-[[:digit:]]{3}-[[:digit:]]{4}$/';  $match = preg\_match($phone\_pattern, $phone); // Returns 1  **testing a date:**  $date = '8/10/209'; // invalid date  $date\_pattern = '/^(0?[1-9]|1[0-2])\/' . '(0?[1-9]|[12][[:digit:]]|3[01])\/' . '[[:digit:]]{4}$/';  $match = preg\_match($date\_pattern, $date); // Returns 0  **A regular Expression Example**: |

**Php try…catch…**

|  |
| --- |
| try { $fv = calculate\_future\_value(10000, 0.06, 0);  echo 'Future value was calculated successfully.'; }  catch (Exception $e) {  echo 'Error: ' . $e->getMessage(); } |

**Php Session:**

|  |
| --- |
| **1: why?**  Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, error messages, etc).  By default, session variables last until the user closes the browser.  **2: use session**  $\_SESSION  session\_start();  unset($\_SESSION['name']); // will delete just the name data  session\_destroy(); // will delete ALL data associated with that user. |

**PHP and SQL:**

|  |
| --- |
| **1:Keywords**    **2: order by**  SELECT columnName1, columnName2,…  FROM tableName  ORDER BY column ASC  //Use ASC for ascending order, Use DESC for descending order.  **3:insert**  INSERT INTO tableName (columnName1, columnName2,…columnNameN)  VALUES (value1, value2, … valueN)  **4:update**  UPDATE tableName  SET columnName1 = value1,  columnName2 = value2,  …  columnNameN = valueN  WHERE criteria |

**Php MVC**

|  |
| --- |
| **Model:**  •In its simplest form the model stores data which is to be accessed by the view and written to by the controller.  •The model is the most complex of all the parts of the system and will contain all the logic which is specific to the application and where domain entities that relate to real world concepts (such as "a user" or "an order") are stored.  •The model takes data (from any source) and processes it. The model also handles all data access and storage. It has no knowledge of any controllers or views which may use it.  **View:**  •The view contains all the display logic.  •In PHP it will be the part of the application which generates the HTML.  •The view can have direct access to the Model and can query the model to get its data.  •The View can create callbacks to its controller (e.g., a clicking a button in the view would trigger an action in the controller).  •Some MVC definitions / instantiations state that the view is decoupled from everything else and fed data by the controller.  **Controller:**  •The controller takes user input and updates the model where required.  •The controller accesses the model, and should does not contain any display code (in the purest form of MVC). All the controller does is respond to user input. |

**Php secure topics:**

|  |
| --- |
| **1: encryption: reversible,**  str\_rot13, xor,base64\_encode/base64\_decode  **methods of encryption**:  symmetric(secret key) encryption, the secret key can never be revealed.  Asymmetric(public key) Encryption. Anyone who knows the secrete key can decrypt the message.  **2: hashing: impossible to decode(non-reversible),the output is always of a fixed length. But different messages can result in the same hash value(collisions are possible)**  md5,sha1 ,password\_hash,password\_verify, **sha256,sha512** are more secure.  hashedmessage = hash(“sha1”, $plaintextmessage);  password\_hash uses **salt** so that your hashes significantly more difficult to crack.  **Store SALT**: When using password\_hash() or crypt(), the return value includes the salt as part of the generated hash. This value should be stored verbatim in your database, as it includes information about the hash function that was used and can then be given directly to password\_verify() or crypt() when verifying passwords.  **3:Extended Validation SSL Certificates(EV)**  **4: SQL injection**  txtUserId = getRequestString("UserId");  txtSQL = "SELECT \* FROM Users WHERE UserId = " + txtUserId;  **bad thing**:  SELECT \* FROM Users WHERE UserId = 105 or 1=1,  **Or even**: we can drop tables as the input  **How can we stop?** : use SQL parameters  SQL parameters are values that are added to an SQL query at execution time, in a controlled manner.  **Eg:**  $stmt = $mysqli->prepare("INSERT INTO CountryLanguage VALUES (?, ?, ?, ?)"); $stmt->bind\_param('sssd', $code, $language, $official, $percent);  $code = 'DEU'; $language = 'Bavarian'; $official = "F"; $percent = 11.2; |