Debugging PHP

Debugging PHP

- In an ideal scenario, PHP development is typically organized into three separate environments:
  - development,
  - staging, and
  - production.
- Each environment serves a specific purpose in the software development lifecycle, providing a structured and controlled progression from code creation to deployment.

Debugging PHP

```
1  // Development Environment Code
2  $debugMode = true;
3  if ($debugMode) {
4          // Display detailed error messages during development
5      ini_set('display_errors', 1);
6      error_reporting(E_ALL);
7  }
8  // Your PHP code for development
9  echo "Hello, Developer!";
10
```

- $debugMode Variable:
  - The $debugMode flag is set to true, commonly serving as a switch to control specific behaviors in the development environment.
  - It determines whether detailed error messages should be displayed during development.
- Error Reporting Configuration:
- Within the if ($debugMode) block:
  - ini_set('display_errors', 1);: Enables the display of error messages on the screen, facilitating quick issue identification and resolution during development.
  - error_reporting(E_ALL);: Sets the error reporting level to E_ALL, ensuring all types of errors (notices, warnings, errors) are reported. This provides comprehensive feedback to developers during the development phase.

Debugging PHP

- Staging Environment:
- Purpose:
  - The staging environment is a replica of the production environment where the application undergoes thorough testing before deployment. It aims to catch issues that might not be apparent in the development environment.
- Characteristics:
  - Mimics the production environment closely.
  - Rigorous testing of new features and updates.
  - Integration testing with other systems or services.

Debugging PHP

```php
// Staging Environment Code
$databaseConfig = [
    'host' => 'staging-db.example.com',
    'username' => 'staging_user',
    'password' => 'staging_password',
    'database' => 'staging_database'
];
// Perform thorough testing in the staging environment
// ...
// Code for connecting to the staging database
$connection = new mysqli($databaseConfig['host'], $databaseConfig['username'], $databaseConfig['password'], $databaseConfig['database']);
if ($connection->connect_error) {
    die("Connection failed: " . $connection->connect_error);
}
echo "Connected to Staging Database!";
```

- In the staging environment, configurations such as database connection details are specific to the staging setup.
- Rigorous testing is performed, and any issues discovered are addressed before moving to the production environment.

Debugging PHP

- Production Environment:
- Purpose:
  - The production environment is the live or public-facing environment where the fully tested and stable version of the application is deployed for end-users.
- Characteristics:
  - High reliability and stability.
  - Optimized configurations for performance.
  - Monitoring tools for real-time performance analysis.
  - Strict security measures.

Debugging PHP

```php
// Production Environment Code
$databaseConfig = [
    'host' => 'production-db.example.com',
    'username' => 'prod_user',
    'password' => 'prod_password',
    'database' => 'prod_database'
];
// Disable detailed error messages for security in production
ini_set('display_errors', 0);
error_reporting(0);
// Code for connecting to the production database
$connection = new mysqli($databaseConfig['host'], $databaseConfig['username'], $databaseConfig['password'], $databaseConfig['database']);
if ($connection->connect_error) {
    die("Connection failed: " . $connection->connect_error);
}
echo "Connected to Production Database!";
```

- In the production environment, configurations are optimized for security and performance.
- Detailed error messages are usually disabled to prevent sensitive information from being exposed.
- Connections to databases and other services are established with production-specific credentials.

Debugging PHP

- Key Benefits of Having Separate Environments:
  - **Isolation**: Each environment serves a distinct purpose, preventing interference between development activities and live production services.
  - **Risk Mitigation:** Issues can be identified and addressed in the staging environment before deployment to production, reducing the risk of introducing bugs or disruptions.
  - Performance Optimization: Configurations in each environment can be tailored to the specific needs of that stage in the development lifecycle, ensuring optimal performance in production.
- Development Workflow:
  - Developers write and test code in the development environment.
  - Code is then pushed to the staging environment for more extensive testing, including user acceptance testing (UAT).
  - Once thoroughly tested and approved, the code is deployed to the production environment for public access.

Debugging PHP

- Outline
  - The PHP.ini Settings
  - Error Handling
    - Error Reporting
    - Exceptions
    - Error Suppression
    - Triggering Errors
    - Error Handlers
    - Error Logs

Debugging PHP

- The php.ini Settings Overview:
  - Environment-wide settings vary based on the server type used for development.
  - Specific configurations are recommended for development, staging, and production environments.

Debugging PHP

- display_errors:
  - Toggle controlling the display of PHP errors.
  - Set to 0 (off) in production to enhance security.
- error_reporting:
  - Defines constants to report errors to the log or browser.
  - Common values: E_ALL (all errors), E_WARNING (warnings), E_DEPRECATED (runtime notices).
- error_log:
  - Specifies the path to the error log file.
  - Records errors in text form, commonly in apache2/logs for Apache servers.
- variables_order:
  - Sets the order of precedence for loading superglobal arrays (e.g., $_GET, $_POST).
- request_order:
  - Describes the order in which PHP registers variables into the $_REQUEST array.
- zend.assertions:
  - Controls whether assertions are run and throw errors.
- assert.exception:
  - Determines whether the exception system is enabled.

Debugging PHP

- Additional Settings:
  - Various settings like ignore_repeated_errors can be used for specific requirements.
  - For instance, it can suppress repeating errors logged from the same line of code.
- Dynamic INI Setting Changes:
  - Some INI settings can be altered during code execution.
  - Caution is advised, especially in production, but it can be useful in the staging environment.
- Example of Dynamic Setting Change:
  - Use error_reporting(E_ALL); and ini_set("display_errors", 1); at the top of a file to override settings temporarily.
  - Helpful for debugging in a controlled environment like staging.

Debugging PHP

- PHP error directives for server environments

PHP directive	Development	Staging	Production
`display_errors`	On	Either setting, depending on desired outcome	Off
`error_reporting`	`E_ALL`	`E_ALL & ~E_WARNING & ~E_DEPRECATED`	`E_ALL & ~E_DEPRECATED & ~E_STRICT`
`error_log`	*/logs* folder	*/logs* folder	*/logs* folder
`variables_order`	EGPCS	GPCS	GPCS
`request_order`	GP	GP	GP

Debugging PHP

- Error Handling
  - Error Reporting
  - Exceptions
  - Error Suppression
  - Triggering Errors
  - Error Handlers
  - Error Logs

Debugging PHP

- **Error Handling**
- Error handling is an important part of any real-world application.
- PHP provides a number of mechanisms that you can use to handle errors, both during the development process and once your application is in a production environment.
  - Error Reporting
  - Exceptions
  - Error Suppression
  - Triggering Errors
  - Error Handlers
  - Error Logs

Debugging PHP

- Error Reporting
  - Normally, when an error occurs in a PHP script, the error message is inserted into the script's output.
  - If the error is fatal, the script execution stops.
- Error Levels:
  - Three levels: Notices, Warnings, and Errors.
  - Notice: May indicate an error or occur during normal execution.
  - Warning: Nonfatal error; script continues execution.
  - Error: Fatal condition, script cannot recover.

Debugging PHP

- Error Reporting
- Script Output:
  - PHP script inserts error messages into output.
  - Fatal errors halt script execution.
- Parse Error:
  - Specific error type for syntactically incorrect scripts.
  - All errors, except parse errors, are runtime errors.
- Handling Recommendations:
  - Treat notices, warnings, and errors equally.
  - Helps prevent issues like using variables before having valid values.
  - By default, all conditions (except runtime notices) are caught and displayed

Debugging PHP

- Error Reporting
- Error Reporting Configuration:
  - Globally configure error reporting in php.ini with error_reporting option.
  - Locally change behavior in a script using error_reporting() function.
- Bitwise Operators:
  - Use bitwise operators to combine constant values for error reporting.
  - Examples:
  - All error-level options: (E_ERROR | E_PARSE | E_CORE_ERROR | E_COMPILE_ERROR | E_USER_ERROR)
  - Exclude runtime notices: (E_ALL & ~E_NOTICE)
- track_errors Option:
  - In php.ini, setting track_errors stores the current error description in $PHP_ERRORMSG.
  - Useful for accessing error details programmatically.

Debugging PHP

- Error-reporting values

Value	Meaning
E_ERROR	Runtime errors
E_WARNING	Runtime warnings
E_PARSE	Compile-time parse errors
E_NOTICE	Runtime notices
E_CORE_ERROR	Errors generated internally by PHP
E_CORE_WARNING	Warnings generated internally by PHP
E_COMPILE_ERROR	Errors generated internally by the Zend scripting engine
E_COMPILE_WARNING	Warnings generated internally by the Zend scripting engine
E_USER_ERROR	Runtime errors generated by a call to `trigger_error()`
E_USER_WARNING	Runtime warnings generated by a call to `trigger_error()`
E_USER_NOTICE	Runtime notices generated by a call to `trigger_error()`
E_ALL	All of the above options

Debugging PHP

- Exceptions
- Exception Usage in PHP Functions:
  - Many PHP functions use exceptions instead of terminating operations.
  - Exceptions allow scripts to continue execution even after an error occurs.
- Exception Handling:
  - When an exception occurs, an object (subclass of BaseException) is created and thrown.
  - Thrown exceptions must be caught by code following the throwing code.

```
1 try {
2   $result = eval($code);
3 } catch (\ParseException $exception) {
4   // Handle the exception
5 }
```

- *Key Points:*
  - *Exception handling allows scripts to gracefully handle errors without abrupt termination.*
  - *try block encloses code where an exception might occur.*
  - *catch block catches and handles the exception.*
  - *Uncaught exceptions halt script execution; hence, handling is crucial.*

Debugging PHP

- Error Suppression
- Error Suppression Operator:
  - Use @ before an expression to disable error messages for that specific expression.
  - Example: $value = @(2 / 0);
- Purpose:
  - Prevents errors from halting script execution.
  - Useful for scenarios where ignoring errors won't compromise the program stat
- Limitation:
  - The error suppression operator cannot trap parse errors; it only works for various runtime errors.

Debugging PHP

- Error Suppression
- Consequences:
  - Suppressing errors means you won't be aware of their occurrence.
  - Handling errors appropriately is generally preferred over suppression.
- Turning Off Error Reporting:
  - To disable error reporting entirely, use error_reporting(0);.
  - Ensures no errors are sent to the client (except parse errors).
- Note:
  - Disabling error reporting doesn't prevent errors; it only prevents their display.
  - Better approaches for controlling displayed error messages involve defining error handlers, as discussed in the "Defining Error Handlers" section.

Debugging PHP

- Triggering Errors
- Purpose of Triggering Errors:
  - Triggering errors manually helps simulate specific conditions for testing and debugging.
- Function for Triggering Errors:
  - Use trigger_error() function to intentionally generate errors during script execution.
- Syntax:
  - ***trigger_error("Custom error message", E_USER_ERR***OR);
- Parameters:
  - First parameter: Custom error message.
  - Second parameter: Specifies the error type, e.g., E_USER_ERROR for a user-generated error.
- Error Types:
  - Common types include E_USER_ERROR, E_USER_WARNING, and E_USER_NOTICE.
  - E_USER_ERROR indicates a fatal error that terminates script execution.

Debugging PHP

- Triggering Errors
- Example:
  - ***function divide($numerator, $denominator) {***
  - ***if ($denominator === 0) {***
  - ***trigger_error("Division by zero is not allowed", E_USER_ERROR);***
  - ***}***
  - ***return $numerator / $denominator;***
  - ***}***
- The primary goal of this code is to handle the scenario where a division by zero is attempted.
- Instead of allowing the division to proceed and result in a mathematical error, the code explicitly triggers a user-generated error using trigger_error.

Debugging PHP

- Error Handlers
  - Error handlers in PHP are mechanisms that allow developers to customize the way errors and exceptions are handled during script execution.
  - By defining custom error handlers, developers can control how errors are logged, displayed, or otherwise managed.
  - The error handler is a function or method that gets called when a PHP runtime error occurs. It can be set using the ***set_error_handler*** function.
  - The custom error handler is invoked whenever an error of the specified type occurs, allowing developers to handle errors in a way that suits their application.

Debugging PHP

- Error Handlers
- Example
  - ***function customErrorHandler($errno, $errstr, $errfile, $errline) {***
  - ***// Custom error handling logic***
  - ***}***
  - ***set_error_handler("customErrorHandler");***
- Parameters passed to the error handler:
  - $errno: The level of the error.
  - $errstr: The error message.
  - $errfile: The file in which the error occurred.
  - $errline: The line number where the error occurred.

Debugging PHP

- Error Logs
  - Error logs are files that record information about errors, warnings, and other relevant messages generated by a software application.
  - In the context of PHP, error logs are particularly useful for debugging and monitoring the health of web applications.
  - PHP allows developers to configure error logging settings to capture different types of messages and store them in a designated log file.

Debugging PHP

- Error Logs
  - **Error Logging Configuration:**
  - The error_log directive in the php.ini file is used to configure the location where PHP errors and other messages are logged.
  - Example in php.ini:
  - ***; Log errors to a file***
  - ***error_log = "/path/to/error_log_file.log"***