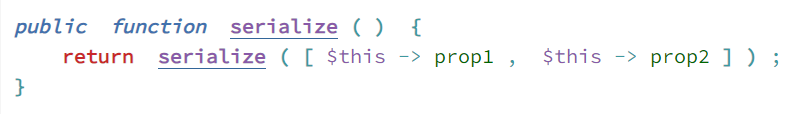
Lưu Anh Sơn

**VIDEO – 2.19**

Trong PHP, "serialize" là quá trình chuyển đổi một đối tượng hoặc một dãy dữ liệu phức tạp thành một chuỗi có thể lưu trữ hoặc truyền qua mạng. PHP cung cấp một số cách để thực hiện serialize đối tượng và dữ liệu. Trong bài viết này, chúng ta sẽ tìm hiểu về cách serialize đối tượng và phương thức serialize magic trong PHP.

Các lớp triển khai Serializablegiao diện được mã hóa bằng Cđịnh dạng, về cơ bản là C:ClassNameLen:"ClassName":PayloadLen:{Payload}, trong đó Payloadlà một chuỗi tùy ý. Đây là chuỗi được trả về Serializable::serialize()và hầu như luôn được tạo ra bởi lệnh gọi lồng nhau tới serialize():



PHP hiện cung cấp hai cơ chế để tuần tự hóa các đối tượng tùy chỉnh: Các phương thức \_\_sleep()/ \_\_wakeup()magic, cũng như Serializablegiao diện. Thật không may, cả hai cách tiếp cận đều có những vấn đề sẽ được thảo luận sau. RFC này đề xuất thêm một cơ chế tuần tự hóa tùy chỉnh mới để tránh những vấn đề này.

**Có thể tuần tự hóa**

Các lớp triển khai Serializablegiao diện được mã hóa bằng Cđịnh dạng, về cơ bản là C:ClassNameLen:"ClassName":PayloadLen:{Payload}, trong đó Payloadlà một chuỗi tùy ý. Đây là chuỗi được trả về Serializable::serialize()và hầu như luôn được tạo ra bởi lệnh gọi lồng nhau tới serialize():

***public***  ***function***  [**serialize**](http://www.php.net/serialize) **( )**  **{**

**return**  [**serialize**](http://www.php.net/serialize) **( [** $this -> prop1 , $this -> prop2 **] )** ;

**}**

Để thể hiện danh tính đối tượng (cùng một đối tượng được sử dụng nhiều lần trong biểu đồ giá trị được tuần tự hóa) và các tham chiếu PHP, định dạng tuần tự hóa sử dụng các tham chiếu ngược đến các giá trị trước đó trong chuỗi được tuần tự hóa. Ví dụ: trong [$obj, $obj]phần tử đầu tiên sẽ được tuần tự hóa như bình thường, trong khi phần tử thứ hai sẽ là phần tử phản hồi có dạng r:1.

Để duy trì danh tính đối tượng (và các tham chiếu) giữa các giá trị là một phần của các đối tượng có thể tuần tự hóa, các serialize()lệnh gọi lồng nhau bên trong Serializable::serialize()chia sẻ trạng thái tuần tự hóa với serialize()lệnh gọi bên ngoài.

Thật không may, điều này có nghĩa là các chuỗi được tạo bởi serialize()các lệnh gọi lồng nhau chỉ hợp lệ nếu chúng cũng không được xác nhận theo thứ tự trong cùng một ngữ cảnh. Một trường hợp đáng chú ý và phổ biến trong đó không phải như vậy là khi người ta cố gắng soạn xê-ri hóa thông qua việc sử dụng parent::serialize():

***lớp*** A ***triển khai*** Serializable **{**

***Private***  $prop ;

***chức năng công khai***  [**tuần tự hóa**](http://www.php.net/serialize) **( ) { trả về** [**tuần tự hóa**](http://www.php.net/serialize) **(** $ this -> prop **)** ; **} *chức năng công khai*** [**unserialize**](http://www.php.net/unserialize) **(** $payload **) {** $this -> prop = [**unserialize**](http://www.php.net/unserialize) **(** $ payload **)** ; **} } *lớp*** B ***mở rộng*** A **{ *private*** $prop ;

***public***  ***function***  [**serialize**](http://www.php.net/serialize) **( )**  **{**

**return**  [**serialize**](http://www.php.net/serialize) **( [** $this -> prop , parent :: [**serialize**](http://www.php.net/serialize) **( ) ] )**

**}**

***public***  ***function***  [**unserialize**](http://www.php.net/unserialize) **(** $payload **)**  **{**

**[** $prop , $parent **]**  = [**unserialize**](http://www.php.net/unserialize) **(** $ payload **)** ;

cha mẹ :: [**hủy xác nhận**](http://www.php.net/unserialize) **(** $parent **)** ;

$cái này -> chống đỡ = $prop ;

**}**

**}**

Mã dạng này không hoạt động đáng tin cậy vì các lệnh gọi serialize()và unserialize()lồng nhau được thực hiện theo các thứ tự khác nhau:

Trong quá trình tuần tự hóa, đầu tiên serialize()lệnh gọi vào A::serialize()được thực hiện và sau đó là lệnh gọi vào B::serialize(). Ngược lại, trong quá trình hủy tuần tự hóa, trước tiên lệnh unserialize()gọi vào B::unserialize()được thực hiện và sau đó là lệnh gọi vào A::unserialize(). Do sự khác biệt trong thứ tự cuộc gọi này, các tham chiếu ngược được tạo trong quá trình tuần tự hóa sẽ không còn chính xác trong quá trình hủy tuần tự hóa.

Vì lý do này, việc sử dụng mã parent::serialize()thường bị hỏng, mặc dù các vấn đề có thể không biểu hiện ngay lập tức (ví dụ vì không có tham chiếu ngược nào được sử dụng).

Vấn đề chính thứ hai Serializablegặp phải là các lệnh gọi phải Serializable::unserialize()được thực hiện *ngay lập tức* khi gặp phải một đối tượng như vậy trong quá trình hủy tuần tự hóa, nếu không việc hủy tuần tự hóa sẽ không xảy ra trong ngữ cảnh chính xác.

Việc thực thi mã tùy ý trong quá trình hủy tuần tự hóa là nguy hiểm và đã dẫn đến nhiều unserialize()lỗ hổng trong quá khứ. Vì lý do này, \_\_wakeup()các cuộc gọi hiện bị trì hoãn cho đến khi kết thúc quá trình hủy đăng ký. Đầu tiên, toàn bộ biểu đồ giá trị được xây dựng và chỉ sau đó tất cả \_\_wakeup()các phương thức xếp hàng mới được gọi.

Điều này khiến chúng ta rơi vào tình huống Serializable::unserialize()được gọi ngay lập tức, trong khi \_\_wakeup()bị trì hoãn. Như vậy, phương thức trước sẽ nhìn thấy các đối tượng trước khi chúng được hủy tuần tự hóa hoàn toàn. Ví dụ: điều này làm cho việc sử dụng DateTimecác đối tượng bên trong Serializable::unserialize()không an toàn vì chúng chưa được khởi tạo đầy đủ.

Vấn đề thứ ba Serializablecó tính chất kỹ thuật hơn. Vì Serializable::serialize()có thể trả về dữ liệu ở định dạng tùy ý nên không có cách chung nào để phân tích các chuỗi được tuần tự hóa. Cơ chế tuần tự hóa của PHP có thể được thực hiện mạnh mẽ hơn nhiều (và có thể nhanh hơn), bằng cách trước tiên thực hiện một bước phát hiện tất cả các tham chiếu ngược được sử dụng. Tuy nhiên, Serializablengăn chặn điều này, vì các tải trọng mà nó tạo ra hoàn toàn không rõ ràng (mặc dù chúng thường tuân theo định dạng tuần tự hóa thông thường).

**\_\_sleep() và \_\_wakeup()**

\_\_sleep()Cơ chế / cũ hơn *về cơ bản*\_\_wakeup() không bị hỏng theo cách vốn có, nó chủ yếu gặp phải các vấn đề về khả năng sử dụng do phạm vi sử dụng hạn hẹp mà nó thực thi. Serializable

Đặc biệt, \_\_sleep()chỉ có thể được sử dụng để loại trừ các thuộc tính khỏi quá trình tuần tự hóa, nhưng sẽ khó sử dụng nếu biểu diễn được tuần tự hóa khác biệt đáng kể so với biểu mẫu trong bộ nhớ (điều này sẽ yêu cầu thêm các thuộc tính giả bổ sung chỉ được sử dụng để tuần tự hóa). Ngoài ra \_\_sleep(), không soạn thảo vì giá trị trả về parent::\_\_sleep()thường không thể sử dụng trực tiếp do hạn chế về khả năng hiển thị.

Tương tự, \_\_wakeup()cũng bị ràng buộc chặt chẽ với ý tưởng rằng trạng thái tuần tự hóa phải được mã hóa trong các thuộc tính. Nếu cách biểu diễn được xê-ri hóa khác biệt đáng kể so với cách biểu diễn trong bộ nhớ thì điều này cũng đòi hỏi phải sử dụng các thuộc tính giả. Không giống như \_\_sleep(), \_\_wakeup()phương thức này có soạn thảo, ở chỗ nó thường an toàn và có ý nghĩa khi gọi parent::\_\_wakeup().

Cơ chế tuần tự hóa được đề xuất cố gắng kết hợp tính tổng quát của Serializablevới cách tiếp cận triển khai của \_\_sleep()/ \_\_wakeup().

Hai phương pháp ma thuật mới được thêm vào:

*// Trả về mảng chứa tất cả các trạng thái cần thiết của đối tượng.*

***hàm công khai*** \_\_serialize **( )** : [**mảng**](http://www.php.net/array) ; *// Khôi phục trạng thái đối tượng từ mảng dữ liệu đã cho.* ***chức năng công khai*** \_\_unserialize **(** [**mảng**](http://www.php.net/array) $data **)** : void ;

Cách sử dụng rất giống với Serializablegiao diện. Từ góc độ thực tế, sự khác biệt chính là thay vì gọi serialize()Inside Serializable::serialize(), bạn trực tiếp trả về dữ liệu cần được tuần tự hóa dưới dạng một mảng.

Ví dụ sau minh họa cách \_\_serialize()/ \_\_unserialize()được sử dụng và cách chúng kết hợp dưới sự kế thừa:

***lớp*** A **{**

***riêng***  $prop\_a ;

***hàm công khai*** \_\_serialize **( )** : [**mảng**](http://www.php.net/array) **{ return [** "prop\_a" => $this -> prop\_a **]** ; **} *hàm công khai*** \_\_unserialize **(** [**mảng**](http://www.php.net/array) $data **) {** $this -> prop\_a = $data **[** "prop\_a" **]** ; **} } *lớp*** B ***mở rộng*** A **{ *riêng*** $prop\_b

;

***hàm công***  ***khai*** \_\_serialize **( )** : [**mảng**](http://www.php.net/array)  **{**

**return**  **[**

"prop\_b" => $this -> prop\_b ,

"parent\_data" => parent :: \_\_serialize **( )** ,

**]** ;

**}**

***hàm công khai*** \_\_unserialize **(** [**mảng**](http://www.php.net/array) $data **) {**

parent :: \_\_unserialize **(** $data **[** "parent\_data" **] )** ; $cái này -> prop\_b

= $data **[** "prop\_b" **]** ;

**}**

**}**

Điều này giải quyết các vấn đề Serializablebằng cách để lại việc tuần tự hóa và hủy tuần tự hóa thực tế cho việc triển khai bộ tuần tự hóa. Điều này có nghĩa là chúng ta không phải chia sẻ trạng thái tuần tự hóa nữa và do đó tránh được các vấn đề liên quan đến thứ tự tham chiếu ngược. Nó cũng cho phép chúng tôi trì hoãn \_\_unserialize()các cuộc gọi đến khi kết thúc quá trình hủy tuần tự hóa.

Các phương thức \_\_serialize()and \_\_unserialize()sử dụng lại Ođịnh dạng tuần tự hóa được sử dụng bởi tuần tự hóa đối tượng thông thường, cũng như \_\_sleep()/ \_\_wakeup(). Điều này có nghĩa là mảng dữ liệu được trả về \_\_serialize()sẽ được lưu trữ như thể nó đại diện cho các thuộc tính đối tượng.

Về nguyên tắc, điều này làm cho các chuỗi hiện có được tuần tự hóa ở Ođịnh dạng có thể tương tác hoàn toàn với cơ chế tuần tự hóa mới, dữ liệu chỉ được cung cấp theo một cách khác (đối với \_\_wakeup()thuộc tính, đối \_\_unserialize()với dưới dạng mảng rõ ràng). Nếu một lớp có cả hai \_\_sleep()và \_\_serialize()thì lớp sau sẽ được ưu tiên hơn. Nếu một lớp có cả hai \_\_wakeup()thì \_\_unserialize()lớp sau sẽ được ưu tiên hơn.

Nếu một lớp thực hiện cả hai Serializablevà \_\_serialize()/ \_\_unserialize(), thì việc tuần tự hóa sẽ thích cơ chế mới hơn, trong khi việc hủy tuần tự hóa có thể sử dụng một trong hai cơ chế đó, tùy thuộc vào việc sử dụng định dạng C(Serializable) hay (\_\_unserialize). ONhư vậy, các chuỗi tuần tự cũ được mã hóa theo Cđịnh dạng vẫn có thể được giải mã, trong khi các chuỗi mới sẽ được tạo ra ở Ođịnh dạng.

RFC này đề xuất bổ sung các phương pháp ma thuật mới, nhưng cũng có thể sử dụng một giao diện thay thế, mặc dù nó sẽ yêu cầu một số cách đặt tên thể dục để tránh RealSerializable.

Đề xuất này sử dụng các phương pháp ma thuật vì hai lý do. Đầu tiên, họ tương tác tốt. \_\_serialize()và \_\_unserialize()có thể được thêm vào một lớp mà không cần lo ngại về tính tương thích: Chúng sẽ được sử dụng trên PHP 7.4 trở lên và bị bỏ qua trên PHP 7.3 trở lên. Thay vào đó, việc sử dụng giao diện yêu cầu nâng cao yêu cầu phiên bản lên PHP 7.4 hoặc xử lý định nghĩa về giao diện sơ khai theo cách tương thích.

Thứ hai, chúng đúng hơn về mặt ngữ nghĩa. Trong PHP tất cả các đối tượng đều có thể tuần tự hóa theo mặc định. Giao Serializablediện là một cách gọi sai theo nghĩa đó, bởi vì một đối tượng không triển khai Serializablecó thể (và thường là) vẫn có thể tuần tự hóa được. Ngược lại, Serializablecó thể được triển khai cụ thể cho mục đích cấm tuần tự hóa, bằng cách đưa ra một ngoại lệ. Các phương thức ma thuật \_\_serialize()và \_\_unserialize()chỉ là những cái móc để tùy chỉnh chức năng tuần tự hóa, chúng không xác định liệu một đối tượng có thể được tuần tự hóa hay không và nói chung, mã sẽ không có lý do gì để kiểm tra sự hiện diện hay vắng mặt của chúng.

**Tạo đối tượng trong \_\_unserialize()**

Một số người đã bày tỏ mong muốn tạo ra \_\_unserialize()một phương thức tĩnh để tạo và trả về đối tượng không được tuần tự hóa (thay vì trước tiên xây dựng đối tượng rồi gọi \_\_unserialize()để khởi tạo nó).

Điều này sẽ cho phép mức độ kiểm soát thậm chí còn cao hơn đối với cơ chế tuần tự hóa, ví dụ: nó sẽ cho phép trả về một đối tượng đã tồn tại từ \_\_unserialize().

Tuy nhiên, việc cho phép điều này một lần nữa sẽ yêu cầu \_\_unserialize()các hàm gọi ngay lập tức (xen kẽ với việc hủy tuần tự hóa) để cung cấp đối tượng cho các tham chiếu ngược, điều này sẽ gây ra một số vấn đề gặp phải Serializable. Như vậy, điều này sẽ không được hỗ trợ.

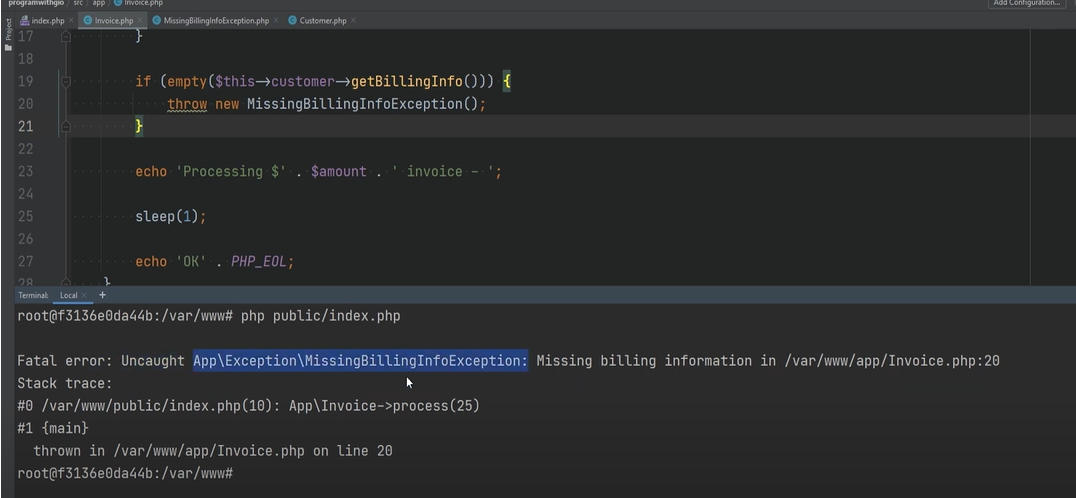
**Những thay đổi không tương thích ngược**

Đề xuất này không có điểm dừng BC. Tuy nhiên, cần lưu ý rằng nó được viết với mục đích không dùng nữa và loại bỏ Serializablegiao diện bị hỏng nghiêm trọng. (Không có nhu cầu cấp thiết cụ thể nào để loại bỏ dần \_\_sleep()và \_\_wakeup().)

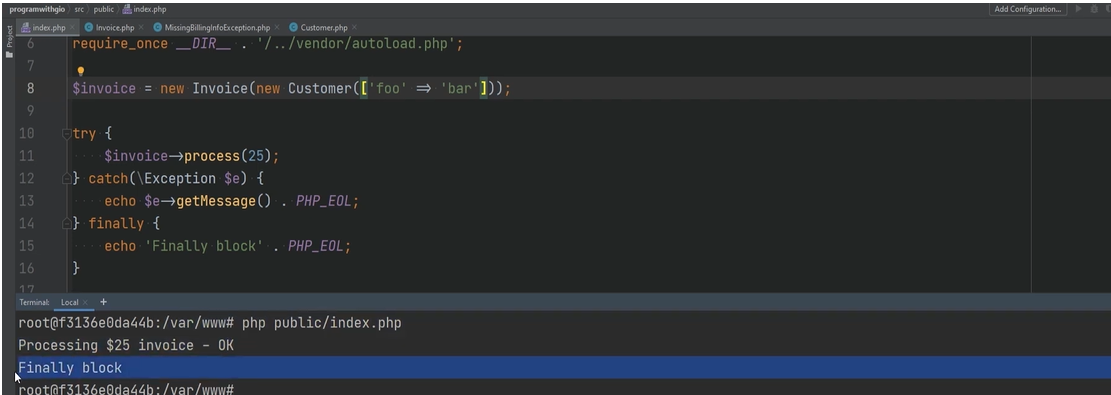
**VIDEO 2.20**

Ngoại lệ chỉ đơn giản là một đối tượng của một số lớp ngoại lệ mô tả một lỗi, nó làm gián đoạn luồng thực thi mã thông thường các ngoại lệ trong PHP có thể được ném theo cách thủ công bằng cách sử dụng từ khóa ném hoặc nó có thể là kết quả của một số loại lỗi trong PHP cho dù nó đến từ các lớp hàm của PHP tích hợp đến từ mã của riêng bạn, bạn chỉ có thể ném ngoại lệ nếu phiên bản của đối tượng được ném là một lớp ngoại lệ.

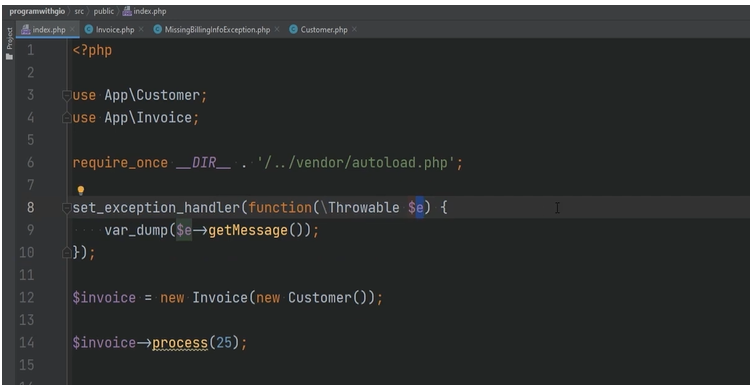
Các ngoại lệ tùy chỉnh, còn được gọi là ngoại lệ do người dùng xác định, là một cách để tạo các lớp ngoại lệ của riêng bạn bằng các ngôn ngữ lập trình hỗ trợ xử lý ngoại lệ. Những ngoại lệ tùy chỉnh này cho phép bạn xử lý các điều kiện lỗi cụ thể có thể phát sinh trong mã của bạn.



Từ lớp ngoại lệ eneus bạn muốn thực thi một số mã bất kể ngoại lệ có được ném hay không, đây là nơi bạn có thể thêm khối cuối cùng để chúng ta có thể xác định cuối cùng



Để đăng ký trình xử lý ngoại elej toàn cục bằng cách sử dụng hàm set\_exception\_handler có chức năng gọi lại tên hàm một bao đóng làm đổi số hàm này ở đây nhận một đối số đơn giản một đối tượng ngoại lệ được ném ra



Các khối thử bắt và bây giờ bạn có thể, vì vậy tại thời điểm này bạn có thể đang hỏi bạn không thể đơn giản trả về một số giá trị mặc định thay vì đưa ra các ngoại lệ

