Web 输入验证系统的对象化设计

Designing for Web Input Validation System based on OODB

杜树杰 （中国海洋大学计算中心 青岛 266003）

摘要：传统意义上从 MSSQL/MySQL 读取并保存数据的过程逻辑上是单向的，对数据的存取没有考虑有效性、合理性等因素。而采用 OOP 方法通过数据库管理数据显然可以使相关系统在逻辑上更合理，数据管理更安全。本文主要介绍如何通过面向对象的方法管理数据库中的记录。作为对象的记录中可以封装数据中蕴含的诸多逻辑，而且其他子类可以继承该逻辑，这样，对象代码中可以实现合法性检验、数据转换和操纵等各类逻辑功能。

关键词：输入验证 OODB 访问方法 重载

1 Web 输入验证系统的设计原则

输入验证是一个很复杂的问题，并且是应用程序开发人员需要解决的首要问题。然而，正确的输入验证是防御目前应用程序攻击的最有效方法之一。正确的输入验证是防止 XSS、SQL 注入、缓冲区溢出和其他输入攻击的有效对策。

输入验证非常复杂，因为对于应用程序之间甚至应用程序内部的输入，其有效构成没有一个统一的答案。同样，对于恶意的输入也没有一个统一的定义。更困难的是，应用程序对如何处理此输入将会影响应用的风险。因此，在对 Web 输入系统进行验证时应遵循以下原则：

（1）假定所有输入都是恶意的。开始输入验证时，首先假定所有输入都是恶意的，除非有证据表明它们并无恶意。无论输入是来自服务、共享文件、用户还是数据库，只要其来源不在可信任的范围之内，就应对输入进行验证。例如，如果调用返回字符串的外部 Web 服务，如何知道该服务不会执行恶意命令呢？另外，如果几个应用程序写入同一个共享数据库，那么在读取数据时，如何知道该数据是否安全呢？

（2）集中方法。将输入验证策略作为应用程序设计的核心元素。考虑集中式验证方法，例如，通过使用共享库中的公共验证和筛选代码。这可确保验证规则应用的一致性。此外，还能减少开发的工作量，且有助于以后的维护工作。

许多情况下，不同的字段要求不同的验证方法，例如，要求使用专门开发的常规表达式。但是，通常可以得到验证常用字段（如电子邮件地址、标题、名称、包括邮政编码在内的通讯地址，等等）的常规方法。

（3）不要依赖于客户端验证。应使用服务器端代码执行其自身的验证。如果攻击者绕过客户端或关闭客户端脚本例程（例如，通过禁用 JavaScript 脚本），后果如何？尽管使用客户端验证可以减少客户端到服务器端的往返次数，但不要依赖这种方法进行安全验证。服务器端的验证才是“深入彻底”的。

（4）注意标准化问题。数据的标准形式是最标准、最简单的形式。标准化是指将数据转化为标准形式的过程。文件路径和 URL 尤其倾向于标准化问题，许多广为人知的漏洞利用都直接源自标准化缺陷。例如，请考虑以下字符串“c: \temp \somefile. dat”，它以标准形式表示了文件及其路径。以下字符串也可以代表同一个文件：“somefile. dat”、“c: \temp \subdir。. \somefile. dat”、“.. \somefile. dat”、“c% 3A% 5Ctemp% 5Csubdir% 5C% 2E% 2E% 5Csomefile. dat”等。

通常，应设法避免让应用程序接受用户输入的文件名，以防止标准化问题。可以考虑其他设计方法。例如让应用程序为用户确定文件名。

如果确实需要用户输入文件名，在作出安全决策（如授予或拒绝对特定文件的访问权限）之前，应确保这些文件名具有严格定义的形式。

（5）限制、拒绝和净化输入。输入验证的首选方

法是从开始就限制允许输入的内容。按照已知的有效类型、模式和范围验证数据要比通过查找已知有害字符的数据验证方法容易。设计应用程序时，应了解应用程序需要输入什么内容。与潜在的恶意输入相比，有效数据的范围通常是更为有限的集合。然而，为了使防御更为彻底，可能还希望拒绝已知的有害输入，然后净化输入。图 1 显示了一种合理的的输入验证策略。

做的优点是：

- 数据库中数据的读写需要通过数据库对象的存取方法（Accessor Methods）来控制
- 在记录和属性级上进行合法性检验
- 关联数据表中对象的存取是"智能化"的
- 对数据库中指定数据的存取都会触发相同的功能代码（即 Accessor Methods），这种逻辑上的可重用性使得系统维护更加方便

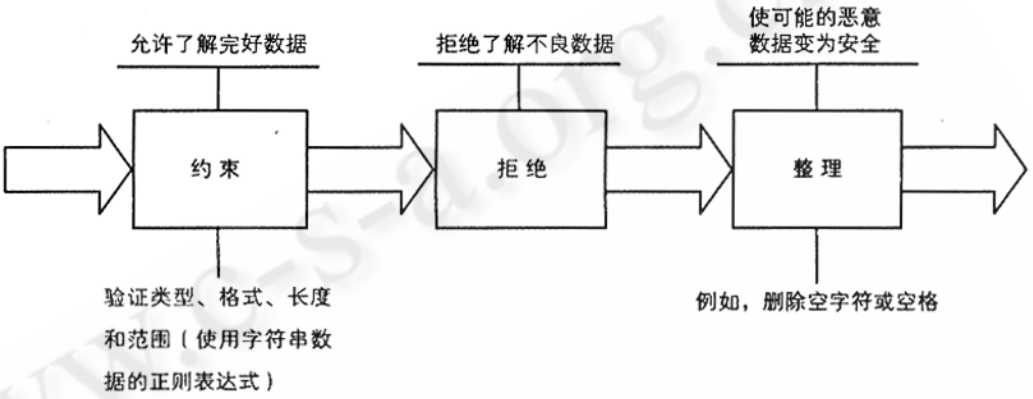

图 1　输入验证策略：限制、拒绝和净化输入

要创建有效的输入验证策略，需了解以下方法及其折衷方案：

- 限制输入。
- 按照类型、长度、格式和范围验证数据。
- 拒绝已知的有害输入。
- 净化输入。

从以上原则可以看出，Web 输入验证过程考虑的因素是多样化的，如果采用传统过程化的代码机制，势必造成系统规模冗长、模块在逻辑上无法独立、系统难以有效扩展等不良后果。鉴于上述原因，将面向对象的数据库技术（OODB）引入输入验证过程对系统的改进应该是一个很好的切入点。

2　基于 OODB 的 PHP 输入验证技术

基于面向数据库对象的输入验证系统设计本质上就是要合理构建一个用户验证类。对于类的构建和使用，传统的 WEB 开发脚本没有提供此项功能。例如，对于一组相关数据来说，传统的 PHP 中使用关联数组（Array 函数）进行处理，而在使用 Zend Engine2 的 PHP5 中则可以构建对象来表述数据库中的数据，这样

- 类中基于记录的逻辑关系更加完善和独立，相比使用各种库文件程序结构更清晰

（1）访问方法（Accessor Methods）的作用。Accessor Methords 可以在类中存取实例变量。例如，对于用户类 User，可以构造 Accessor Method "user→username()"和"user→setusername()"分别返回和设置实例变量（假设为 $username）。

```
<? php
class User {
    var $_data = array(); // User 类中所有与属性相关的数据
    function username() {
      return ! empty( $this - >_data['username']) ? $this - >_data['username'] : '(no name!)';
    }
    function setUsername( $newUsername) {
      if ( $this - > validateUsername ( $newUsername)) {
          $this - >_data['username'] = $newUsername;
```

```
        }
    }
    function validateUsername(&$ someName) {
        if (strlen( $ someName) > 12) {
            throw new Exception('Your username is too long');
        }
        return true;
    }
}
? >
```

采用 Accessor Methods 访问实例变量可以使开发人员能够更灵活地选择用户类的构建和工作方式，同时不影响其他使用该类的代码执行。从上述代码中可以看出使用访问方法的优点：

• 如果 username 值为空，访问方法"username"提供了缺省值。

• 访问方法 setUsername( ) 用以检验 username 赋值的有效性。

这样，程序在访问对象属性时可以进行更好的控制。开发者不可以直接存取 username 属性，这样通过使用 Accessor Method 所有涉及 user 类的代码无需作任何修改就可以实现有效性检查。

由于 username 有效性检查的代码独立于 setUsername( ) 方法，因此将对象存入数据库之前进行有效性检查是容易实现的，其实这种实现方法很适合重载(Override)父类中的特定方法去构建子类，通过重写父类中已经包含的同名函数实现类的派生与多态。如果对象方法实现代码少、功能专一，那么其可重用性就强；如果对象方法的实现规模较为庞大，而且是多目的性的，那么开发者很难不加修改地将父类代码直接应用于子类中。

例如，假设 Admin 是 user 的一个子类，显然对于 Admin 用户在检查口令有效性时应该使用更"苛刻"的方法，最好的实现显然不是重写 setUsername( ) 方法，而是对 user 类中有效性方法的重载。

(2) 基于 Accessor Methods 的系统完善。下面的代码是基于 Accessor Methods 对输入验证系统的完善：任何存取的数据都是经过口令核实的，而不是从关联数组返回静态数据。另外，Accessor Methods 可以修改缓冲数值，因为所有的修改都必须通过 setX( ) 方法，通过该方法可以将依赖于 X 的缓冲数值复位。上面的 User 类在层级上的修改如下：

对于 $ _data 变量的内部操作改为通过私有方法_getData( ) 和_setData( ) 实现。那些非特殊的方法都移入 User 类的抽象超类 Record 中。Record 类处理存取 $ _data 数组的过程，包括修改属性时的 Validation 方法，向负责存储对象记录的实例过程发出修改请求等。

```
<? php
class User extends Record {
    ……
//Record 是所有数据库对象的超类.
abstract class Record {
    var $ _data = array( );
    var $ _modifiedKeys = array( ); // 跟踪哪些字段自创建/存取后修改过
    ……
    function _getData( $ attributeName) {
        return $ this - >_data[ $ attributeName];
    }
    function _setData( $ attributeName, $ value) {
        if ( $ this - > validateAttribute ( $ attributeName, $ value)) {
            if ( $ value ! = $ this - > _data [ $ attributeName]) {
                $ this - > _data [ $ attributeName] = $ value;
                $ this - > _modifiedKeys [ ] = $ attributeName;
                $ this - >didChange( );
            } else {
                // the new value is identical to the current one
                // no change necessary
            }
        }
    }
    // 对于属性"X"，会自动寻找并调用方法"validateX( )"
```

```
    function validateAttribute ( $attributeName, &
$value) {
        $methodName = 'validate'. $attributeName;
        if (method_exists( $this, $methodName)) {
            return $this -> $methodName( $value);
        } else {
            return true;
        }
    }
    function didChange( ) {
        // 通知"对象存储类"记录已经改变
    }
}
.....
? >
```

可以将某些代码从 User 类中移到超类 Record 中，User 子类只需考虑与用户相关的项目，如 accessors 和 validation 方法，而通过 SQL 对数据库记录的交互操作全部放在负责对象存储的类中，因此该类还应承担其子类 Record 的实例化。这样就使得 Record 类在规模上不再庞大，效率也会提高。这点在处理多种对象时是很重要的因素。

3 结束语

安全性应渗透在产品开发生命周期的所有阶段，还应成为应用程序设计的关键问题。而输入验证过程是可靠的身份验证和授权策略设计的重要一环。目前，大多数应用程序级攻击都源于恶意形式的输入数据和薄弱的应用程序输入验证设计。本文提出的基于对象数据库的输入验证系统设计思想以 Accessor Methods 为中心，使得系统的安全性、扩展性和系统化程度大大提高，在逻辑设计上更加独立和开放，从设计方法上提出了解决构建安全应用程序的新思路。

参考文献

1 Havard Lindset , PHP, MySQL and Authentication , www. devarticles. com, 2002,7,7.

2 Matt Wade, Database Abstraction with PEAR, codewalkers. com, 2003,9,6.

3 Vladmir Krstulja, User identification using cookies in PHP/MySQL, www. devarticles. com, 2003,3,28.

4 Jesus Castagnetto, Professional PHP Programming, Wrox Press, 2001,3.