عنوان: **چرا به الگوهای طراحی JavaScript نیاز داریم ؟**
نویسنده: شاهین کیاست
تاریخ: ۱۳۹۱/۰۳/۳۰ ۲۱:۵۵
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, JavaScript

امروزه اهمیت یادگیری JavaScript بر هیچ کس پوشیده نیست ، APIهای جدید HTML 5 و امکانات جدید وب مثل Geo Location Offline Web applications ، Canvas ، Offline storage ، و ... همه نیاز به دانش JavaScript دارند. در نتیجه نیاز است علاوه بر یادگیری JavaScript به تمیز کد نوشتن جهت سهولت نگهداری آگاه بود. همانطور که در کدهای سمت سرور مثل C# و یا PHP نیاز به استفاده از الگوهای طراحی (Design Patterns)است در JavaScript هم اوضاع به همین منوال است.
الگوی طراحی یک راه حل قابل استفاده مجدد است که برای حل مشکلات متداول در طراحی نرم افزار به کار می‌رود.
چرا به الگوی‌های طراحی JavaScript نیازمندیم ؟
می خواهیم کد هایی با قابلیت استفاده‌ی مجدد بنویسیم ، استفاده از عملکردهای مشابه در سطح صفحات یک Web application یا چند Web Application.
می خواهیم کد هایی با قابلیت نگهداری بنویسیم ، هر چه قدر در فاز توسعه کدهای با کیفیت بنویسیم در فاز نگهداری از آن بهره می‌بریم. باید کد هایی بنویسیم که قابل Debug و خواندن توسط دیگر افراد تیم باشند.
کدهای ما نباید با توابع و متغیرهای دیگر پلاگین‌ها تداخل نامگزاری داشته باشند. در برنامه‌های امروزی بسیار مرسوم است که از پلاگین‌های Third party استفاده شود. می‌خواهیم با رعایت Encapsulation and modularization در کدهایمان از این تداخل جلوگیری کنیم.

معمولا کدهای JavaScript که توسط اکثر ما نوشته می‌شود یک سری تابع پشت سرهم هست ، بدون هیچ کپسوله سازی :

```
function getDate() {
    var now = new Date();
    var utc = now.getTime() + (now.getTimezoneOffset() * 60000);
    var est;
    est = new Date(utc + (3600000 * -4));
    return dateFormat(est, "dddd, mmmm dS, yyyy, h:MM:ss TT") + " EST";

}
function initiate_geolocationToTextbox() {
    navigator.geolocation.getCurrentPosition(handle_geolocation_queryToTextBox);
}

function handle_geolocation_queryToTextBox(position) {
    var longitude = position.coords.longitude;
    var latitude = position.coords.latitude;
    $("#IncidentLocation").val(latitude + " " + longitude);
}
```

به این روش کدنویسی Function Spaghetti Code گفته می‌شود که معایبی دارد :
توابع و متغیرها به Global scope برنامه افزوده می‌شوند.
کد Modular نیست.
احتمال رخ دادن Conflict در اسامی متغیرها و توابع بالا می‌رود.
نگهداری کد به مرور زمان سخت می‌شود.

با شبیه سازی یک مثال مشکلات احتمالی را بررسی می‌کنیم :

```
// file1.js
function saveState(obj) {
    // write code here to saveState of some object
    alert('file1 saveState');
}
// file2.js (remote team or some third party scripts)
function saveState(obj, obj2) {
     // further code...
    alert('file2 saveState");
}
```

همانطور که می‌بینید در این مثال در 2 فایل متفاوت در برنامه مان از 2 تابع با اسامی یکسان و امضای متفاوت استفاده کرده ایم .
اگر فایل‌ها را اینگونه در برنامه آدرس دهی کنیم :

```
<script src="file1.js" type="text/javascript"></script>
<script src="file2.js" type="text/javascript"></script>
```

متد saveState در فایلی که دیرتر آدرس داده شده (file2.js) ، متد saveState در file1.js را Override می‌کند ، در نتیجه عملکردی که از متد saveState در فایل اول انتظار داریم اتفاق نمی‌افتد.
در پست بعدی به راه حل این مشکلات و کپسوله سازی خواهم پرداخت.
برای مطالعه‌ی بیشتر کتاب (Learning JavaScript Design Patterns) را از دست ندهید.

نظرات خوانندگان

نویسنده: NargesM
تاریخ: ۱۳۹۱/۰۳/۳۱ ۶:۸

سلام.. ممنون