37 | 包管理和分发：通过NPM做包的管理和分发

2022-12-13 石川 来自北京

《JavaScript进阶实战课》

课程介绍 >

讲述：石川

时长 12:52　大小 11.76M

你好，我是石川。

在前面几讲中，我们看到无论是响应式编程框架 React，还是测试用的 Jest、Puppeteer 工具，亦或是做代码检查和样式优化的 ESLint、Prettier 工具，都离不开第三方库，而我们在之前的例子中，都是通过 NPM 下载和安装工具的。

所以，今天我们就来深入了解下 NPM 以及包的管理和发布。

包的发布

NPM（Node Package Manager）虽然它叫做 Node 包管理，但是其实你也可以用它管理或发布用 JavaScript 以外的语言编写的程序的包，只不过 NPM 最主要的受众还是 JavaScript 在 Web 和服务器端的开发者。在 NPM 中，有两个核心的概念，一个是包（package），另外一个是模块（module）。

包是一个含有 package.json 文件的文件夹。package.json 的作用是对包中的文件或目录的描述。一个包必须含有一个 package.json 文件才能发布到 NPM 的注册列表。

模块是任何可以被 Node.js 的 require() 加载的文件或目录。能够被成功加载的模块必须是一个带有 main 字段的 pakcage.json 的目录，或者一个 JavaScript 的文件。

一个包的发布很简单，首先在命令行通过创建 mkdir 和改变目录 cd 的命令，我们可以创建一个包的文件夹，并导航到包的根目录。在目录下，我们可以创建一个 package.json 的文件。package.js 文件的创建方式有两种，一种是直接创建，另外一种是在命令行上执行 npm init 的命令，通过提示输入后，生成 package.json 的文件。

在一个 package.json 中，必须要包含的字段是名称和版本号。有些时候，一个包是带有对其它包的依赖的，在这种情况下，在 package.json 中，也可以设置相关的依赖关系。

如果模块是在 Git 上管理的，可以在包所在的根目录下，将模块加进来。或者，我们也可以选择直接在目录下写一个模块。以下面的 printMsg 模块为例，这里通过 `export` 导出的模块，可以在其它的程序中通过 `require()` 导入引用。

复制代码

```
exports.printMsg = function() {
  console.log("This is a message from the demo package");
}
```

在我们创建了一个包之后，在正式发布前，最好先通过 `npm install` 自己测试一下。确保无误之后，我们可以通过 `npm publish` 对包进行发布。**无论是公开还是私有包的发布，都需要在发布前在 NPM 的注册页面上创建一个用户。**为了安全，发布包以前，最好通过 2FA 的双因子认证。

以上是对公开包的发布流程。除了公开的包以外，我们也可以发布私有的包。那么公开和私有的包有什么区别呢？对于公开的包来说，所有人都有读取和下载的权限，针对指定的用户或组织，可以设置写和发布的权限。对于私有的包来说，只有指定的人员或组织内的成员才可以读取、下载、写入或发布相关的包。

包的持续集成和部署

在 DevOps，我们经常强调 CI/CD 的持续集成和部署。在使用 NPM 的时候，我们也可以通过访问令牌的方式来代替用户名和密码的认证方式。这里的访问令牌是一个十六进制字符串，我们可以使用它来进行身份验证、安装或发布模块。通过这种方式，我们可以让其它工具，例如持续集成测试环境访问 NPM 的包。当我们的工作流在运行时，它可以完成包括安装私有包的任务。

而令牌又分为两类，一种是**传统令牌**，另外一种是**粒度令牌**。其中，传统令牌主要分为 3 类，一种是只读的，它只可以下载包；第二种是在下载的基础上可以安装；第三种是在前两种的基础上可以发布包。但是从安全的角度考虑，粒度令牌，顾名思义，有更细粒度的权限管理。粒度令牌还可以更好地区分可以访问的包和范围，授权给指定的组织，设置过期时间，基于 CIDR 的方法来控制授权的 IP 范围，并且提供只读和读写的权限选项。

举个例子，我们可以通过下面的方式，创建一个基于 IP 范围的访问令牌。

复制代码

```
npm token create --cidr=192.0.2.0/24
```

之后，我们可以将令牌设置为 CI/CD 服务器中的环境变量或密钥。例如，在 GitHub Actions 中，我们可以将令牌添加为密钥。然后根据该密钥，创建一个名为 NPM_TOKEN 的环境变量，将密钥提供给工作流。

复制代码

```
steps:
  - run: |
      npm install
  - env:
      NPM_TOKEN: ${{ secrets.NPM_TOKEN }}
```

这里值得强调的是，这里的令牌具有可以读取私有包，代表我们发布新包，更改用户或包设置的权限。所以从安全的角度考虑，我们必须保护好令牌。千万不能将令牌添加到版本控制或存在不安全的地方。最好是将令牌存储在密码管理器、云提供商的安全存储或 CI/CD 工具提供的安全存储器中。如果可能，应该使用我们前面讲到的具有最低权限的粒度访问令牌，并为令牌设置较短的过期时间。

包的管理

前面，我们说过，对于包的权限，我们可以选择公开或者私有化。无论是哪种模式下，我们都可以对用户和组织进行管理。区别只是在于，公开的包所有人都可以读取和下载；而私有的包，无论是读取下载还是写入发布，都需要指定的用户或组织，才能赋予权限。下面，我们就来看看，如何在 NPM 中对组织进行管理。

首先，我们在登录后，点击头像可以选择创建一个组织，之后，我们需要给组织起一个名字。这个时候，我们可以选择免费和付费的方式，这两者的区别在于免费的版本可以让我们创建公开的包，而私有的包则需要付费。在这个过程中，你也可以选择将一个个人账户转换成一个组织账户。

这时，我们也可以通过姓名和邮箱邀请加入组织的用户。我们可以选择用户的角色和加到的团队，比如角色可以是“成员”或者是“管理员”。在这里，只有所有者有权利添加或删除组织中的成员，改变成员的角色，或者对组织的名称做修改或者删除。“管理员”可以管理团队，这里包括了团队的创建和删除，同时可以管理团队成员的加入和移除，以及包的访问权限。“成员”的权限是可以创建、发布组织范围内的包。

在组织创建的时候，一个“开发”组是自动生成的。除此之外，组织的所有者和管理员也可以创建更多的组。比如对于有些用户，我们不希望他们拥有开发权限，可以将他们从“开发”组移到项目管理组。

安全考虑

前面，我们学习了 NPM 的发布、CI/CD 和管理的流程。下面，我们再来看看 NPM 的安全考虑。在使用 NPM 的时候，很重要的一点就是安全考量。这里，我们可以从开发者和使用者两个不同的角度来看。

首先，我们先**从开发者的角度**来看，这里最需要警惕的就是我们的账户安全。

针对子账户安全，最容易受到的就是密码攻击。当然密码攻击是一种常见的网络攻击，不仅限于 NPM，而是在任何 Web 服务上都有可能遭受到的攻击。保护帐户安全的最佳方法就是我们前面提到的启用双因子身份验证（2FA）。

在此基础上，安全性最强的选项是使用安全密钥，无论是内置于设备还是外部硬件的密钥。安全密钥可以将身份验证绑定到正在访问的站点，大大**降低网络钓鱼的风险**。但因为并不是所有人都可以访问到安全密钥，所以 NPM 还支持为 2FA 生成一次性密码的身份验证应用程序。

由于这种攻击的常见性，以及 NPM 包的流行程度和对开源生态系统的影响，NPM 采取了分期的方法，对排名前 100 的软件包维护者和排名前 500 的软件包管理者强制实行 2FA 的认证。当然，这样的举措还是远远不够的，在不久的将来，所有高影响力软件包的维护者（每周下载量超过 100 万次或依赖 500 次以上的软件包）都将被强制执行 2FA 验证。如果你作为包的发布者不选择 2FA 的验证，NPM 会通过向你的电子邮箱发送一次性密码来加强登录验证，以防止帐户被盗取。

盗取帐户的另一种方法是，通过**使用过期域名作为电子邮件地址来识别帐户**。攻击者可以注册过期的域名并重新创建用于注册帐户的电子邮件地址。通过访问帐户的注册电子邮件地址，攻击者可以通过重置密码来盗取不受 2FA 保护的帐户。

当一个包在发布的时候，发布包与帐户关联的电子邮件地址是包含在公共元数据中的。攻击者能够利用这些公共数据来识别可能容易被账户盗取的账户。NPM 也会定期检查帐户电子邮件地址是否有过期的域名或无效的 MX 记录。域名在过期之后，NPM 会禁用帐户进行密码重置，并要求用户在重置密码之前进行帐户恢复或成功通过身份验证的流程。需要注意的是，作为包的维护者，在你更新电子邮箱地址的时候，存储在包的公共元数据中的电子邮箱地址是不会更新的。由于这种抓取公共元数据以识别易受过期域名影响的帐户将导致误报，因此这些帐户看似易受攻击，但实际上并非如此。

下面，我们再**从使用者的角度**来看看。攻击者可能会试图通过注册与流行软件包名称相似的软件包来诱骗他人安装恶意软件包，希望人们因为笔误输入错误的名称，或者用其它的方式混淆两者。NPM 能够检测错别字攻击并阻止这些包的发布。这种攻击的一种衍生攻击是当公共包与组织正在使用的私有包以相同的名称注册时，私有包被公共注册表中的包取代。所以这里的一个建议是，使用范围包（scoped package)，以确保私有包不会被公共注册表中的包取代。

另外一个问题是对现有包的恶意更改行为，在这里，攻击者也有可能不会诱骗用户使用类似名称的软件包，而是试图将恶意行为添加到现有的流行软件包中。为了解决这个问题，NPM 也在与微软合作，扫描软件包中已知的恶意内容，并运行软件包来寻找潜在的新的恶意行为模式。这使得 NPM 包中带有恶意内容的比例大幅减少。此外，NPM 的信任和安全团队会检查并删除用户报告的恶意行为和内容。

总结

通过这一讲，我们看到了在 JavaScript 不断模块化的今天，NPM 让我们更容易分享和使用其他开发者所提供的工具，而且除了模块自身的功能外，NPM 也可以是很好的版本和依赖管理

工具，并且 NPM 的使用也可以促进团队的协作。但是同时，我们也看到了它的很多安全隐患。所以在使用的时候，还是要谨慎。作为开发者，我们不希望自己好心提供的工具成为了黑客进行攻击的手段，同时，作为用户，我们更应该注意管理从 NPM 下载的第三方代码的风险。

思考题

前面，我们讲了在使用 NPM 的时候，最重要的就是安全的考量，当然除了我们上面说到的这些方法外，之前在安全的一讲中，我们提到的漏扫也都是降低风险的方式。那么在实际开发中，你会使用 NPM 吗？除了我们讲到的方法外，你还能想到其他的安全措施吗？

欢迎在留言区分享你的经验、交流学习心得或者提出问题，如果觉得有收获，也欢迎你把今天的内容分享给更多的朋友。我们下节课再见！

上一篇 36丨Flow：通过Flow类看JS的类型检查

下一篇 38丨编译和打包：通过Webpack、Babel做编译和打包

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。