JavaScript 逆向爬虫程序设计

邢泽浩 胡逸同

目录

1. 网站常用的数据保护方法

现代网站面临着越来越多的数据安全威胁，因此采取有效的数据保护方法至关重要。常用的网站数据保护方法涵盖了多个方面，旨在确保数据的机密性、完整性和可用性。

1.1 JavaScript 的压缩和混淆

JavaScript
的压缩和混淆是网站常用的数据保护方法，旨在提高网站的代码安全性和保护知识产权。通过压缩和混淆 JavaScript
代码，可以减少代码的文件大小，使代码难以理解和逆向工程，从而降低恶意用户或攻击者对代码的分析和篡改风险。通过 JavaScript
压缩。我们可以消除 JavaScript 代码中的空格、注释和其他无关字符来减小文件大小。这有助于提高网页加载速度，并减少网络传输的时间和带宽。常见的
JavaScript 压缩工具包括 UglifyJS、Terser 等，它们能够自动压缩代码并删除不必要的字符。通过
JavaScript
混淆，我们可以通过重命名变量、函数和字符串，使其变得无意义和难以理解。混淆过程中，变量和函数名被替换为随机的、无关的名称，字符串常量也经过转换或拆分处理。这样做不仅增加了代码的复杂性，使其对外部观察者更加难以解读，还增加了逆向工程的难度。

变量命名与函数命名

变量命名和函数命名是 JavaScript 代码混淆的关键步骤之一。通过将代码中的变量名和函数名替换为无意义的随机字符，可以大大增加代码的复杂性和难以理解性，从而阻碍对代码的逆向分析和理解。

在变量命名方面，常见的做法是使用工具如 UglifyJS、JavaScript
Obfuscator 等来对代码进行处理。这些工具使用算法和策略将原始的变量名转换为随机生成的字符。通过这种方式，变量的语义和用途对于外部观察者来说变得模糊不清，增加了理解代码逻辑的难度。

类似地，函数命名也可以经过混淆处理。函数名是代码中的重要标识符，通过将其替换为无意义的字符，如 f1、f2、f3
等，可以使函数的用途和功能对外部人员更加难以推断。这种混淆技术不仅使函数名变得晦涩难懂，还可以掩盖代码的执行流程，增加对代码结构的理解难度。

值得注意的是，变量命名和函数命名的混淆不仅仅是简单的替换字符。混淆工具通常会使用更复杂的算法和策略来生成随机字符，以增加代码的混乱性和可预测性。常见的方法有对称加密，非对称加密，哈希函数等，依靠
AES，RSA，SHA-256 等算法来实现命名的混淆。此外，这些工具还会考虑到代码的语法规则和上下文，以确保混淆后的代码仍然是有效的
JavaScript 代码。

总而言之，通过将变量名和函数名替换为无意义的随机字符，可以大幅增加 JavaScript
代码的复杂性和难以理解性。这种混淆技术有效地阻碍了对代码的逆向分析和理解，提高了代码的安全性和保护知识产权的能力。

1.2 字符串与常量混淆

字符串和常量混淆是 JavaScript 代码混淆的重要手段。通过对代码中的字符串和常量进行混淆处理，可以增加代码的复杂性和混乱性，使其对外部观察者更加难以理解和还原。

一种常见的字符串混淆技术是将字符串拆分为多个字符，并将其存储为一个数组或变量。例如，将字符串"Hello"拆分为['H', 'e', 'l', '
l', 'o']，然后将其存储为一个变量，如 var str = ['H', 'e', 'l', 'l', '
o']。在代码执行时，可以通过拼接数组元素来还原原始的字符串，如 str.join('')，得到"Hello"。

另一种常见的混淆技术是使用 Unicode 编码来表示字符串和常量。通过将字符转换为其对应的 Unicode 编码，如将字母'A'
转换为'\u0041'
，可以使字符串在源代码中呈现为一系列看似无意义的字符，增加了对字符串含义的解读难度。

此外，还可以对字符串和常量进行动态解码。通过使用算法或函数，在运行时动态生成解码逻辑，以将混淆的字符串还原为其原始值。这种动态解码的方式使代码更加复杂和混乱，增加了对字符串和常量的理解难度。

字符串和常量混淆具有很高的安全性：字符串和常量混淆并非只是简单的替换或拆分字符。混淆工具会使用更复杂的算法和策略来生成混淆后的字符串和常量，以增加代码的混乱性和安全性。

通过字符串和常量混淆，JavaScript
代码的可读性和可理解性大大降低，使其对外部观察者和潜在攻击者更加难以理解和逆向分析。这种混淆技术有助于保护代码的机密性和知识产权，并提高网站的安全性。然而，需要权衡混淆带来的代码复杂性对性能的影响，以及对代码维护和调试的困难程度。

1.3 常量计算和抽象化

常量计算和抽象化是 JavaScript 代码混淆中的另一项重要技术。通过对常量的计算和抽象化，可以将代码中的具体常量值隐藏起来，增加代码的复杂性和难度，使其对外部观察者更加难以理解和还原。

常量计算是指通过对常量进行数学运算或逻辑操作，将其替换为计算结果。例如，假设有一个常量声明为 const MAX_VALUE =
1000，通过对其进行计算，例如 MAX_VALUE * 2，可以将其替换为 2000。这样做不仅隐藏了具体的常量值，还增加了代码的复杂性和理解难度，因为外部观察者无法直接获得常量的真实值。

抽象化是指将具体的常量值替换为更通用和抽象的形式。例如，将常量值替换为变量或其他常量，使其不再具有明确的含义。例如，将常量 PI
的值 3.1415926 抽象化为一个变量，如 const
PI = Math.random()。这样做增加了代码的混乱性和难以理解性，使常量的含义更加模糊。

常量计算和抽象化的目的是增加代码的复杂性，使其对外部观察者更加难以理解和还原。这种混淆技术可以防止恶意用户或攻击者轻易地获取代码中的具体常量值，从而增加代码的安全性和保护知识产权的能力。

然而，需要注意的是，常量计算和抽象化虽然能够增加代码的混乱性，但也可能增加代码的复杂性和运行时的性能开销。因此，在进行常量计算和抽象化时需要权衡安全性和代码性能之间的平衡，并确保代码仍然具有可维护性和可读性。

综上所述，常量计算和抽象化是 JavaScript
代码混淆的一种重要技术。通过隐藏常量的具体值和抽象化常量的含义，可以增加代码的复杂性和难度，从而提高代码的安全性和保护知识产权的能力。然而，混淆过程需要注意权衡安全性和代码性能，并确保代码的可维护性和可读性。

1.4 混淆代码结构

常量计算和抽象化是 JavaScript
代码混淆的重要手段之一，旨在增加代码的复杂性和难度，以提高代码的安全性和保护知识产权的能力。通过对常量的计算和抽象化处理，可以有效隐藏代码中的具体常量值，使其对外部观察者更加难以理解和还原。

常量计算的方法是对常量进行数学运算或逻辑操作，将其替换为计算结果。例如，将常量值 4 加上常量值 6，可以通过计算得到
10，并将其作为新的常量值。这样做不仅隐藏了具体的常量值，还增加了代码的复杂性和理解难度，使外部观察者无法轻易获取常量的真实值。

另一方面，常量抽象化是将具体的常量值替换为更通用和抽象的形式。通过将常量值转换为变量或其他抽象的表示形式，如将常量值 3.14
抽象为常量 PI，可以增加代码的混乱性和难以理解性。这种抽象化处理使常量的具体含义变得模糊，增加了代码的复杂性和逆向分析的难度。

常量计算和抽象化的目的是增加代码的复杂性和混乱性，使其对外部观察者更加难以理解和还原。这种混淆技术在保护敏感数据、算法逻辑和商业机密等方面起到了重要作用。通过隐藏常量的具体值，代码的机密性得到了加强，减少了恶意用户或攻击者对代码的分析和篡改的风险。

常量计算和抽象化是 JavaScript
代码混淆中的重要手段，通过隐藏常量的具体值和抽象化常量的含义，增加了代码的复杂性和难度，提高了代码的安全性和保护知识产权的能力。这种混淆技术在保护敏感数据和防止逆向工程方面具有重要作用，但需要权衡安全性和代码性能，以确保代码的可维护性和可读性。

1.5 动态代码生成

动态代码生成是 JavaScript 代码混淆中的一项关键技术，它能够在运行时动态生成代码，从而增加代码的复杂性和难度，使其对外部观察者更加难以分析和理解。

一种常见的动态代码生成技术是使用 eval()函数。eval()
函数允许将一个字符串作为 JavaScript 代码进行解析和执行。通过将代码片段存储为字符串，然后在运行时使用 eval()
函数来执行它，可以动态生成代码并避免在静态代码中直接暴露代码逻辑。这种技术使得代码的执行流程更加难以预测和分析，增加了代码的混乱性和安全性。

另一种常见的动态代码生成技术是使用 Function()构造函数。Function()
构造函数可以接受字符串形式的参数作为函数体，并在运行时创建一个新的函数对象。通过构建函数体的字符串，可以动态生成函数，包括变量、逻辑操作和控制流程等。这种方式使得代码的结构和行为更加动态化，增加了代码的复杂性和难度，使外部观察者更加难以分析代码的逻辑。

字符串拼接也是一种常见的动态代码生成技术。通过将代码片段拆分为多个字符串，然后在运行时将它们拼接起来并执行，可以动态生成代码。这种方式可以将代码的结构和逻辑隐藏在多个字符串中，使其对外部观察者更加难以理解和还原。

1.2 对敏感数据进行加密

网站对敏感数据进行加密是一种常见的数据保护方法，它能够保护数据在传输和存储过程中的安全性。通过加密敏感数据，即使在数据被非法访问或窃取的情况下，攻击者也无法轻易获得明文数据。

通过对敏感数据进行加密，即使在数据泄露或被未授权访问的情况下，攻击者也无法直接获得可读的明文数据。加密提供了一层额外的安全保护，确保数据的机密性和完整性。然而，需要注意的是，密钥的生成和管理、加密算法的选择以及安全的存储和传输等方面需要特别注意，以确保加密系统的安全性。这里介绍几种典型的加密方式，介绍其中的原理和方法。

1.2.1 对称加密：AES

对称加密是一种加密算法，使用相同的密钥进行加密和解密操作。在对称加密中，发送方和接收方共享一个密钥，用于加密和解密数据。AES
是其中的典型算法

AES（Advanced Encryption Standard）是一种对称加密算法，广泛应用于数据加密和保护领域。以下是使用 AES 对称加密算法进行加密和解密的基本步骤：

加密步骤：

1. 选择 AES 加密算法的密钥长度（通常为 128、192 或 256 位）。
2. 根据选择的密钥长度生成一个随机的密钥。
3. 将明文数据分块（通常为 128 位一块），并对每一块进行以下操作： a. 使用密钥和 AES 加密算法进行加密。每一块的加密操作都依赖前一块的加密结果。
   b. 对加密后的结果进行一定的填充操作，确保每个数据块的长度一致。
4. 最后一块可能需要额外的填充操作，以满足加密算法的要求。

解密步骤：

1. 使用相同的密钥和 AES 解密算法，对密文数据块进行解密操作。
2. 对解密结果进行逆向填充操作，恢复原始的明文数据块。
3. 对每个解密的数据块进行逆向操作，恢复原始的明文数据。
4. 合并所有明文数据块，得到最终的解密结果。

需要注意的是，在使用 AES 进行加密和解密时，密钥的安全性非常重要。密钥应该保持机密并且只能被授权的人员访问。

在实际的编程中，可以使用各种编程语言提供的加密库或框架来实现 AES 加密和解密操作。例如，Java 中可以使用 javax.crypto 包提供的
AES 加密功能，Python 中可以使用 PyCryptodome 或 cryptography 等库实现 AES 加密功能。

请注意，在实施加密算法时，应遵循最佳的安全实践，并考虑其他因素，如数据完整性、密钥管理、随机数生成等，以确保加密方案的安全性。

1.2.2 非对称加密 RSA

非对称加密是一种加密算法，使用不同的密钥进行加密和解密操作。在非对称加密中，包括公钥和私钥两个密钥，公钥用于加密数据，私钥用于解密数据。RSA
是非对称加密算中最常用的算法之一

RSA（Rivest-Shamir-Adleman）是一种非对称加密算法，广泛应用于数据加密、数字签名和密钥交换等领域。以下是使用 RSA
非对称加密算法进行加密和解密的基本步骤：

1. 密钥生成： a. 选择两个不同的大素数 p 和 q。 b. 计算 n = p * q，这是 RSA 算法中的公共模数。 c. 计算欧拉函数 φ(n) = (
   p-1) * (q-1)。 d. 选择一个整数 e，1 < e < φ(n)，且 e 和 φ(n) 互质。e 称为公共指数。 e. 计算私钥 d，使得 (e * d) % φ(
   n) = 1。d 称为私钥或私有指数。
2. 加密： a. 将明文数据分成合适大小的数据块，并转换为整数表示。 b. 对每个数据块执行加密操作：c = (m^e) % n，其中 m 是明文数据块，c
   是加密后的密文数据块。

3. 解密： a. 对每个密文数据块执行解密操作：m = (c^d) % n，其中 c 是密文数据块，m 是解密后的明文数据块。

需要注意的是，RSA 加密算法中的公钥是 (e, n)，私钥是 (d, n)。公钥用于加密数据，私钥用于解密数据。

在实际应用中，常用的密钥长度为 1024 位或 2048 位，以提供足够的安全性。

在编程实现上，可以使用各种编程语言提供的加密库或框架来实现 RSA 加密和解密操作。例如，Java 中可以使用 javax.crypto 包提供的
RSA 功能，Python 中可以使用 PyCryptodome 或 cryptography 等库实现 RSA 加密功能。

需要注意的是，RSA 加密算法的性能相对较慢，因此通常用于加密较小量的数据，而不适合加密大型数据。

1.2.3 哈希函数：SHA-256

哈希函数本质上并不是加密算法，而是一种用于加密的单向函数，用于将任意长度的输入数据转换为固定长度的哈希值。

SHA-256（Secure Hash Algorithm 256-bit）是一种哈希函数，用于将任意长度的数据映射为固定长度的哈希值，通常为 256 位（32
字节）。SHA-256 是 SHA-2 哈希函数系列中的一员，提供了更高的安全性和抗碰撞能力。

使用 SHA-256 哈希函数进行数据哈希的基本步骤如下：

1. 准备待哈希的数据，可以是任意长度的二进制数据。
2. 初始化 SHA-256 算法的内部状态。
3. 将数据按照指定的规则进行分块处理。
4. 对每个数据块进行以下操作：
    1. 将数据块与当前的哈希值进行混合。
    2. 根据 SHA-256 算法的规则，对数据块进行一系列的逻辑运算，包括位运算、逻辑函数和模运算等。
    3. 更新当前的哈希值。
5. 当所有数据块处理完成后，得到最终的哈希值，即 SHA-256 哈希结果。

SHA-256 哈希函数的特点包括：

• 单向性：对于给定的输入，很容易计算出对应的哈希值，但从哈希值反推原始输入几乎是不可能的。

• 唯一性：不同的输入很难产生相同的哈希值，且即使输入的稍微变化，哈希值也会有较大的差异。

• 抗碰撞能力：极低的概率使得两个不同的输入产生相同的哈希值。

SHA-256 常用于数据完整性校验、密码存储、数字签名等应用场景，用于验证数据的完整性和防止篡改。

在编程实现上，各种编程语言都提供了 SHA-256 哈希函数的库或模块，可以直接调用这些函数来计算数据的 SHA-256 哈希值。例如，在
Python 中可以使用 hashlib 模块的 sha256() 函数来计算 SHA-256 哈希值。

需要注意的是，SHA-256 是一种单向哈希函数，它只提供了数据的哈希值，无法从哈希值反推出原始数据。因此，在存储密码等敏感信息时，通常使用哈希值而不是明文进行比对。

1.3 访问控制和身份验证

互联网数据保护是当今数字化时代至关重要的一环。随着大量的个人和敏感信息存储在互联网上，确保只有经过授权的用户可以访问这些数据变得至关重要。在数据保护方法中，访问控制和身份验证是关键措施，旨在限制对敏感数据的访问权限，并确保只有合法用户才能获取和操作这些数据。

访问控制机制是数据保护的基石之一。它通过管理用户对系统、资源或数据的访问权限来确保数据的安全性。访问控制依赖于身份验证和授权过程。首先，用户需要进行身份验证以证明其真实身份。这通常涉及使用用户名和密码进行登录。用户名作为唯一标识符，密码作为身份验证的凭据，用户必须提供正确的密码来证明其合法性。

然而，仅凭密码并不足以确保数据的安全。这就引入了多因素身份验证的概念。多因素身份验证要求用户提供两个或更多的独立身份验证因素，例如密码、指纹、短信验证码或硬件令牌。这种额外的验证层增加了安全性，即使密码被泄露，攻击者也需要其他因素才能成功通过身份验证。

一旦用户身份得到验证，访问控制机制将授予用户相应的权限。这通常基于角色和权限控制。角色是一组预定义的权限集合，用户被分配到适当的角色中。每个角色都具有特定的权限，定义了用户能够执行的操作和访问的资源。这种分层授权的方式使得管理用户权限变得更加灵活和可维护。

此外，审计和日志记录也是访问控制的重要组成部分。审计和日志记录记录和监视用户对系统和数据的访问活动。通过记录用户行为，可以追踪和分析用户的活动，及时检测异常行为和安全事件。审计日志可用于后续调查和取证，帮助保护数据的完整性和隐私。

在访问控制和身份验证中，还有一些其他的安全措施。例如，会话管理技术用于跟踪和管理用户在系统中的会话活动。通过控制用户的登录和注销过程，限制会话超时时间，并监测和终止异常或未经授权的会话，可以减少会话劫持和未授权访问的风险。

总而言之，访问控制和身份验证是互联网数据保护方法中的关键要素。它们通过限制对敏感数据的访问权限，确保只有经过授权的用户才能获取和操作这些数据。访问控制机制依赖于身份验证和授权过程，以确保用户的真实身份和相应的权限。

1.4 安全存储和运输

安全存储和运输是互联网数据保护的关键环节之一。它涉及采取一系列措施，以确保数据在存储和传输过程中的安全性。以下是对安全存储和运输常用手段的进一步扩展：

1.4.1 加密传输协议

采用加密传输协议是保护数据在互联网传输过程中的重要手段。其中最常见的是 HTTPS（超文本传输安全协议）。通过使用
HTTPS，数据在客户端和服务器之间的传输过程中会进行加密，从而防止中间人攻击和窃听，确保数据的机密性和完整性。HTTPS
使用公开密钥基础设施（PKI）来验证服务器的身份，并在传输过程中对数据进行加密和解密。

1.4.2 安全的数据库和文件传输系统

为了保护数据的存储安全，采用安全的数据库和文件传输系统至关重要。这包括使用具有强大安全功能的数据库管理系统（DBMS），以及通过安全文件传输协议（如
SFTP）进行文件传输。安全的数据库系统提供访问控制、加密、审计日志等功能，确保只有经过授权的用户可以访问和操作数据。安全的文件传输系统通过加密数据传输和使用身份验证机制，防止未经授权的访问和数据泄露。

1.4.3 防火墙和入侵检测系统

防火墙和入侵检测系统是网络安全的重要组成部分，用于防止外部攻击者对系统进行入侵。防火墙可以监控和过滤网络流量，根据预定义的规则允许或阻止特定类型的数据传输。它可以阻挡恶意入侵和未经授权的访问，提供额外的安全层。入侵检测系统（IDS）可以监测和识别可能的攻击行为和异常活动，及时发出警报并采取相应的应对措施，以保护数据的安全。

综上所述，安全存储和运输是确保互联网数据保护的关键措施。通过采用加密传输协议、安全的数据库和文件传输系统，以及使用防火墙和入侵检测系统来防止外界攻击，可以有效保护数据的机密性、完整性和可用性。这些措施的综合应用有助于构建一个安全可靠的数据存储和传输环境，保护用户隐私和敏感信息的安全。

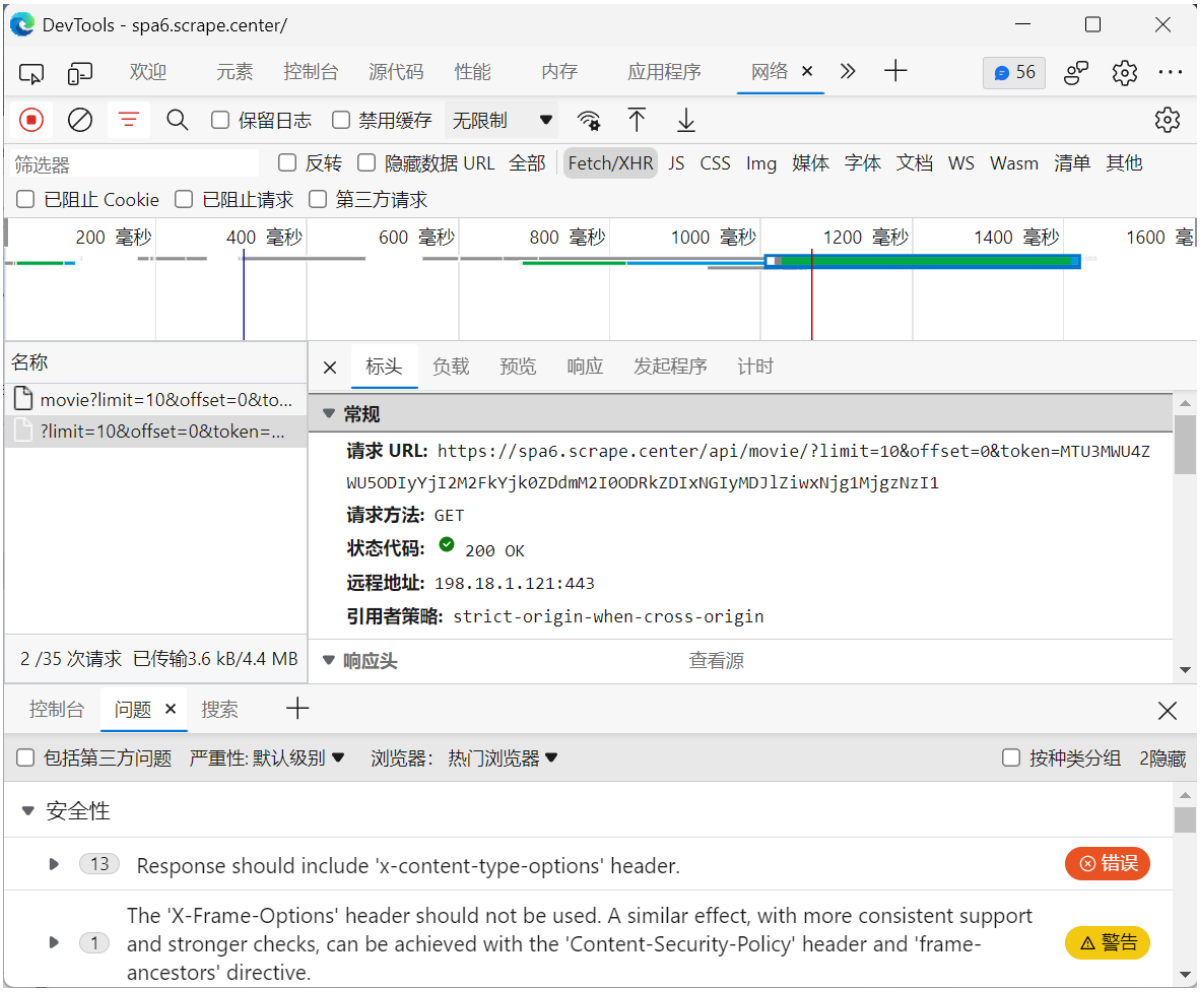
1.5 输入验证和过滤

输入验证和过滤是互联网数据保护中的关键措施，旨在防止恶意用户输入特殊字符或脚本，从而避免安全漏洞和攻击。其中，防跨站脚本攻击
(xSS) 和防止 SQL 注入攻击是常用的手段。跨站脚本攻击是通过注入恶意脚本代码来利用网站漏洞，攻击者可以获取用户敏感信息或进行欺骗。
为了防止这种攻击，必须对用户输入的数据进行验证和过滤，移除或转义特殊字符、标签和脚本，确保用户输入的数据不会被误解释为可执行的代码。
另外，SQL 注入攻击是利用应用程序对用户输入的 SQL 语句未正确过滤和验证而导致的安全漏洞。为了防止这种攻击，应使用参数化查询或预处理语句来构建
SQL 查询，并对用户输入的数据进行过滤，删除或转义可能被误解为 SQL
代码的特殊字符。通过输入验证和过滤，可以有效防止恶意用户注入特殊字符或脚本来利用系统漏洞进行攻击。为了提高效果，应采用最新的安全技术和工具，定期更新和审查安全策略，并进行安全测试和漏洞扫描，以保护系统免受新型攻击的威胁。通过综合应用这些措施，可以有效减少安全漏洞的风险，确保用户输入的数据的合法性和安全性。

2. 分析目标网站的加密方式

2.1 URL 分析

我们开始从网站本身的信息进行入手：打开网站，打开检查，查询 URL 与请求信息。



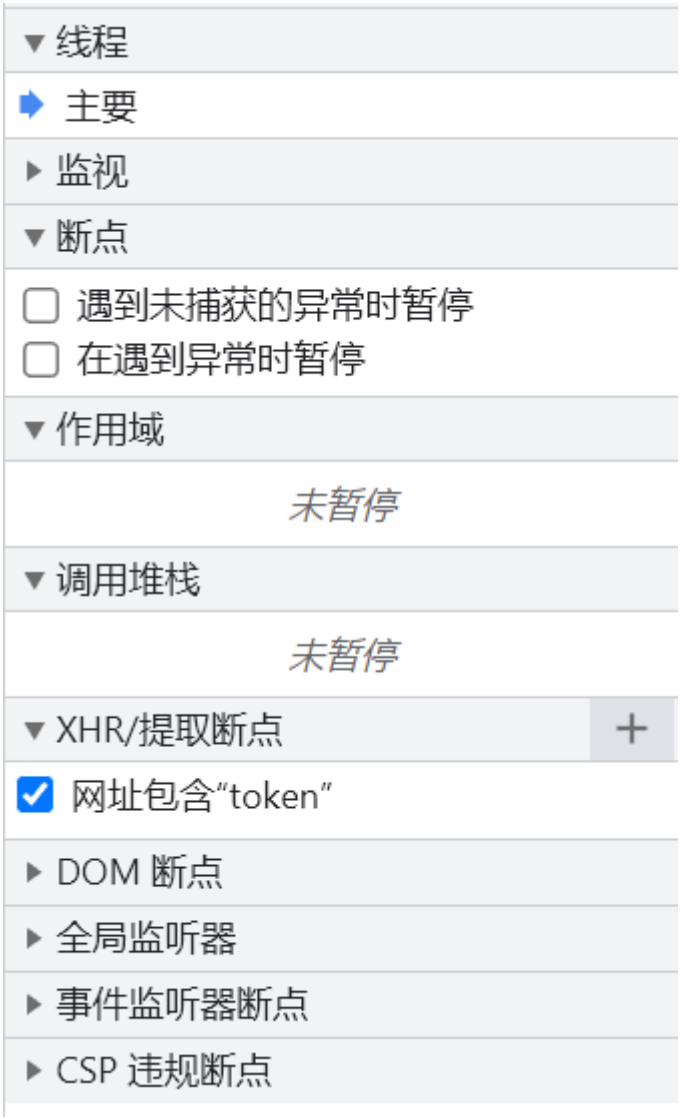
上图为网站首页的 URL 信息，我们发现请求网址中含有 token 数值。

点击进入电影详情界面，发现电影的 request URL 中不仅有 token 数值，还多出来一个加密 ID。

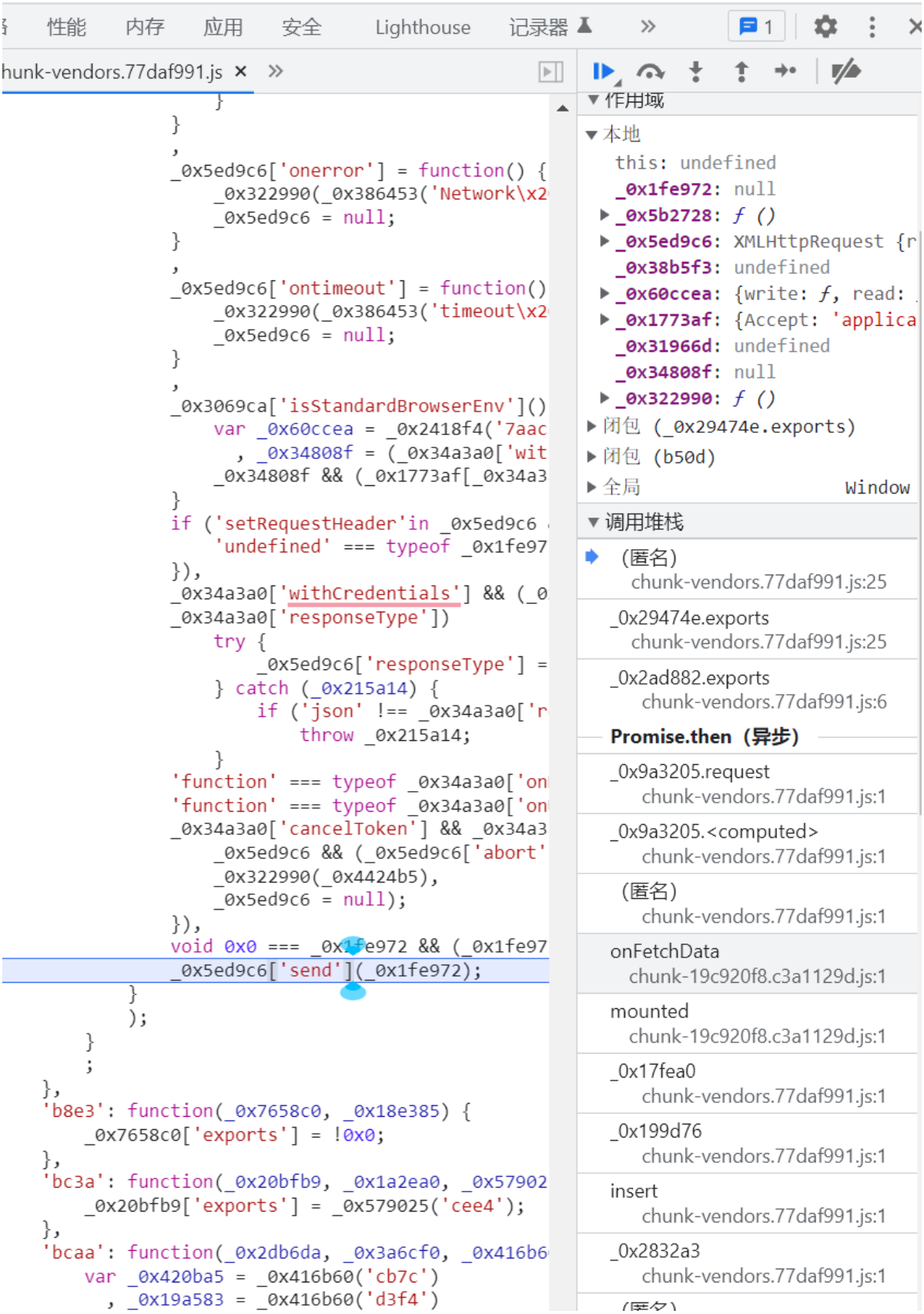
于是我们决断：token 有可能存在于界面的 URL 中，获取 token 可以从页面的 URL 获取中入手，我们可以寻找潜在的 URL 获取参数或传参方法。

2.2 JS 代码分析获取网站 token

于是，我们决定对 JS 代码中与 URL 有关的信息进行查找。我们对‘token’字样进行搜查与断点提取，使用 XHR/提取断点方法：



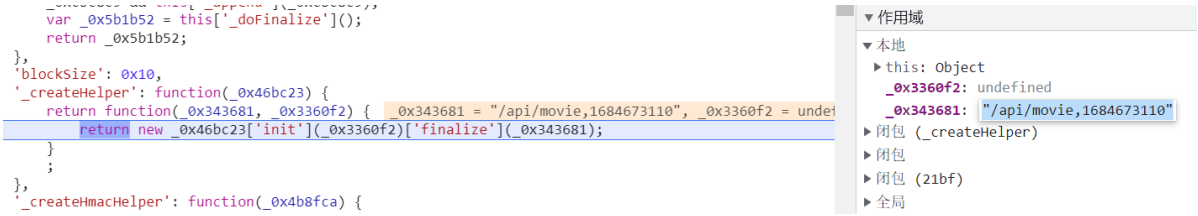
查找到一个名为'cancelToken'的方法中，方法中含有 send 方法，我们认为其中可能存在相关的信息，于是在 send 函数处断下断点重新进行加载：



在 send 方法下断点后，我们在右侧栈中，对代码进行了排查，寻找是否有可能含有 URL 字段值的参数或方法，最后我们找到了
onFetchData 函数，其中很有可能包含对 URL 数值的传递方法。

继续步进方法，我们在接下来的函数中找到 Date 字样，疑似获取了时间戳，并查看到
了‘SHA’、‘Base64’等相关加密字符，很可能与加密有关，这段代码很可能就是加密函数的位置所在：

继续进行逐步的步进检查，发现方法中确实获取了时间戳，还使用 argument 方法获取了网站 URL 的 path，并通过逗号与时间戳拼接在了一起。



发现方法使用了 SHA1 和 Base64 对字符串进行了编码，而方法中有对时间戳的两次链接，经过分析，我们获取了 token 的编码方法：

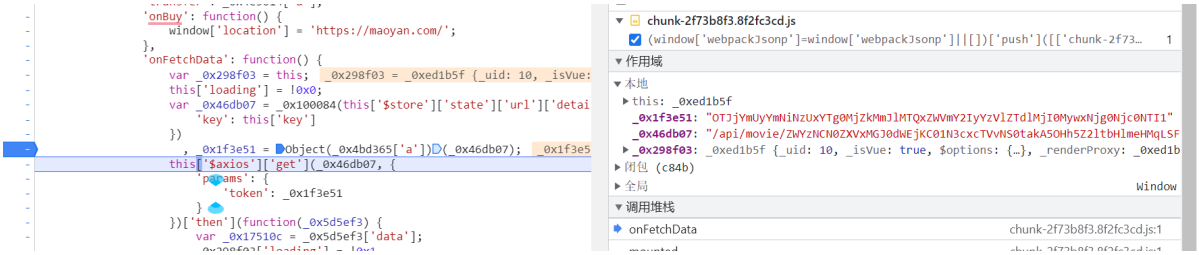
```
x=SHA1(url/api/movie,TimeStamp)
token=Base64(x,TimeStamp)
```

同样的方法，我们获取了详情页面的 token 获取方法：

```
X=SHA1(url/api/movie/encryptID,TimeStamp)
token=Base64(X,TimeStamp)
```

2.3 JS 代码分析获取网站加密 ID

在电影界面的详情页中，我们发现需要获取电影详情页的 token，而这一段 token 的获取可能需要涉及到网站页面的加密 ID，需要从 URL
开始进行寻找：



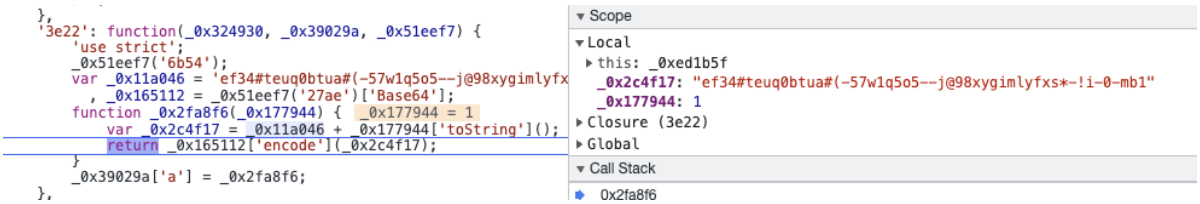
直接查看 URL，搜索 token，读取堆栈，检查到 onFetchData 函数：

```
        , _0x4e3614 = _0x16e3fa('3e22')
        , _0x100084 = _0x16e3fa('1a7b')
        , _0x531a42 = {
            'name': 'Detail',
            'data': function() {
                return {
                    'loading': !0x1,
                    'key': this['$route']['params']['key'],
                    'movie': null
                };
            },
            'mounted': function() {
                this['onFetchData']();
            },
            'computed': {
                'photos': {
```

步进了之后我们找到了 Key 的变量值，但是通过排查该变量所在的函数，我们并没有找到来源于 key 参数的数值或传参方法，于是我们采用暴力解决的方法：我们对
key 进行了全局的搜索，寻找相关的参数传递方法并进行了逐一排查。

```
                    'sm': 0x8
                }
            }, [_0x563af1('router-link', {
                'attrs': {
                    'to': {
                        'name': 'detail',
                        'params': {
                            'key': _0x38ed69['transfer'](_0x38ed69['movie']['id'])
                        }
                    }
                }
            }, [_0x563af1('img', {
                'staticClass': 'cover',
                'attrs': {
                    'src': _0x38ed69['movie']['cover']
                }
            })])], 0x1), _0x563af1('el-col', {
                'staticClass': 'p-h',
```

最后成功找到了 key 的参数传递方法如上，我们设置断点并步进：



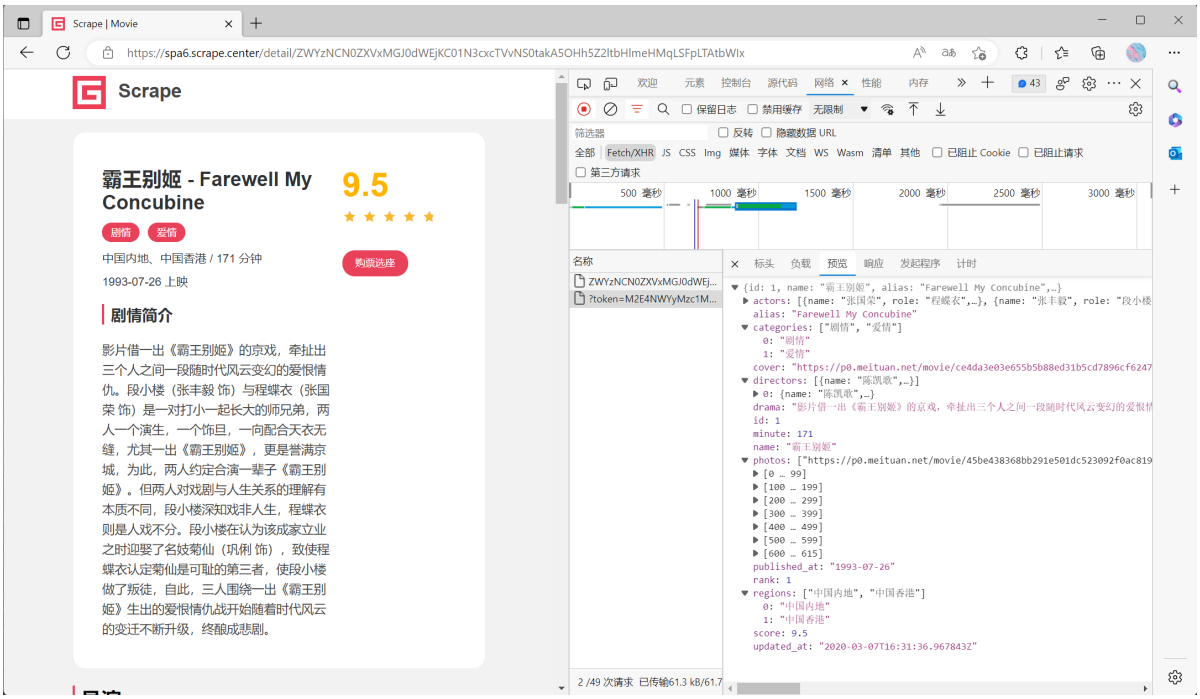
找到了 encode 方法，与 Base64 的相关加密方法。最后通过解析方法我们成功找到了 encryptID 的加密方式，为一串加密字符与 ID
变量进行拼接之后，进行了一次 Base64 加密。加密的规则具体用公式表示为：

```
encryptID=Base64("ef34#teuqObtua#(-57w1q5o5-j@98xygimlyfxs\*-!i-O-mb"+t)
```

3. 爬虫程序设计

3.1 目标

调用网站数据接口，获取电影详情页信息。



3.2 分析

1. 该电影网站的数据通过 Ajax 加载，电影详情页数据接口 (URL) 的返回值 (JSON) 包含指定电影的全部信息，与电影详情页对应，如上图的右侧所示。该接口有如下结构：

> https://spa6.scrape.center/api/movie/{encrypt_id}/?token={token}
>
> - **protocol:** https
> - **host:** spa6.scrape.center
> - **path:** `encrypt_id` 代表经过**加密**的电影唯一标识符（下称电影的原始唯一标识符为电影 id）
> - **query:** URL 设有过期策略，这意味着 `token` 具有**时效性**

2. 网站还使用了电影列表的数据接口，该接口的返回值同样是 JSON 格式，包含了指定电影数量和偏移量的**电影 id** 和电影摘要等数据。
例如 query = ?limit=**2**&offset=0&token=NTU...x 的返回值如下：

```
{
  "count": 101,
  "results": [
    {
      "id": 1,
      "name": "霸王别姬",
      "alias": "Farewell My Concubine",
```

```
      "cover": "https://p0.meituan.net/...c",
      "categories": ["剧情", "爱情"],
      "published_at": "1993-07-26",
      "minute": 171,
      "score": 9.5,
      "regions": ["中国内地", "中国香港"]
    },
    {
      "id": 2,
      "name": "这个杀手不太冷",
      "alias": "Léon",
      "cover": "https://p1.meituan.net/movie/...c",
      "categories": ["剧情", "动作", "犯罪"],
      "published_at": "1994-09-14",
      "minute": 110,
      "score": 9.5,
      "regions": ["法国"]
    }
  ]
}
```


该接口 (URL) 结构如下：

> https://spa6.scrape.center/api/movie/?limit={limit}&offset={offset}&token={token}
>
> **query** 字段有三个参数，分别是：
>
> - `limit`：列表中电影数量
> - `offset`：偏移量
> - `token`：列表页 token

3. 该网站对 JavaScript 代码进行了**混淆**

因此，我们需要根据 JavaScript 代码逆向解析出 `encrypt_id` 和 `token` 的生成方法（见 token 和 加密 id 生成原理），并实现，然后再调用数据接口获取电影详情页信息。

3.3 实现

以下内容采用 notebook 的形式展示。

3.3.1 准备工作

导入依赖库并禁用 SSL 警告

```
import base64
import hashlib
import random
import time
from typing import List, Any

import requests
import urllib3

urllib3.disable_warnings()
```

定义常量

- `INDEX_URL`：电影列表 URL
  - `limit`：电影数量
  - `offset`：偏移量
  - `token`：列表页 token
- `DETAIL_URL`：电影详情 URL
  - `encrypt_id`：电影 id（加密后）
  - `token`：详情页 token
- `SECRET`：密钥（对应 加密id生成原理）

```
INDEX_URL = 'https://spa6.scrape.center/api/movie?limit={limit}&offset={offset}&token={token}'
DETAIL_URL = 'https://spa6.scrape.center/api/movie/{encrypt_id}?token={token}'
SECRET = 'ef34#teuq0btua#(-57w1q5o5--j@98xygimlyfxs*-!i-0-mb'
```

3.3.2 获取 token

- 参数：path (e.g. **/api/movie**)
- 返回：`token`（base64 编码后的字符串）

```
def get_token(args: List[Any]):
    # 获取时间戳
    timestamp = str(int(time.time()))
    # 将时间戳加入参数列表
    args.append(timestamp)
    sign1 = ','.join(args)
    print('1. 拼接 path 和时间戳：\t\t', sign1)
    # 将参数列表转为字符串并进行SHA1加密
    sign2 = hashlib.sha1(sign1.encode('utf-8')).hexdigest()
    print('2. 对上一步结果进行 SHA1 加密：\t', sign2)
```

```
    # 将加密后的字符串和时间戳拼接并进行base64编码
    sign3 = sign2 + ',' + timestamp
    print('3. 拼接加密后的字符串和时间戳: \t', sign3)
    token = base64.b64encode(sign3.encode('utf-8')).decode('utf-8')
    print('4. 对上一步结果进行 base64 编码: ', token, '\n')
    return token
```

使用示例

```
token = get_token(args=['/api/movie'])
print('token:', token)
```

运行结果：

```
1. 拼接 path 和时间戳:          /api/movie,1685265861
2. 对上一步结果进行 SHA1 加密:      5203a412bc9595544c1a98aa342a2987072fafbf
3. 拼接加密后的字符串和时间戳:
5203a412bc9595544c1a98aa342a2987072fafbf,1685265861
4. 对上一步结果进行 base64 编码:
NTIwM2E0MTJiYzk1OTU1NDRjMWE5OGFhMzQyYTI5ODcwNzJmYWZiZiwxNjg1MjY1ODYx

token: 'NTIwM2E0MTJiYzk1OTU1NDRjMWE5OGFhMzQyYTI5ODcwNzJmYWZiZiwxNjg1MjY1ODYx'
```

3.3.3 获取电影列表（Requests 实现）

- 参数：无
- 返回：JSON 格式的电影列表

方法

1. 获取 `token`
2. 构造 URL：limit=10, offset=0, token=`token`
3. 获取 URL 的 response， 提取电影总数
4. 根据电影总数构造新的 URL：limit=`mov_count`, offset=0, token=`token`
5. 接收新的 URL 的 response， 提取 JSON 格式的电影列表
6. 返回电影列表

```
def get_mov_list():
    # 获取token
    token = get_token(args=['/api/movie'])
    # 构造URL
    url = INDEX_URL.format(limit=10, offset=0, token=token)
    # 获取电影总数
    mov_count = requests.get(url, verify=False).json()['count']
    # 根据电影总数构造新的URL
    url = INDEX_URL.format(limit=mov_count, offset=0, token=token)
    # 获取电影列表
    mov_list = requests.get(url, verify=False).json()
    return mov_list
```

使用示例

```
mov_list = get_mov_list()
print(mov_list)
```

运行结果：


```
{
  "count": 101,
  "results": [
    {
      "id": 1,
      "name": "霸王别姬",
      "alias": "Farewell My Concubine",
      "cover": "https://p0.meituan.net/...c",
      "categories": [
        "剧情",
        "爱情"
      ],
      "published_at": "1993-07-26",
      "minute": 171,
      "score": 9.5,
      "regions": [
        "中国内地",
        "中国香港"
      ]
    },
    {
      "...": "..."
    },
    {
      "id": 104,
      "name": "value",
      "alias": null,
      "cover": null,
      "categories": null,
      "published_at": null,
      "minute": null,
      "score": null,
      "regions": null
    }
  ]
}
```


3.3.4 获取电影列表（Playwright 实现）

Playwright 是微软在 2020 年初开源的新一代自动化测试工具，它的功能类似于 Selenium, Pyppeteer 等，可以驱动浏览器进行各种自动化操作。它对市面上的主流浏览器都提供了支持，功能简洁又强大。

我们通过 Playwright 的事件监听方法拦截了 Ajax 请求，直接获取了响应结果。即使这个 Ajax 请求有加密参数 (
e.g. `encrypt_id`, `token`)，我们也不用关心，因为我们直接截获了 Ajax 最后响应的结果。

与 Requests 实现相比，Playwright 实现的代码更加简洁，而且不用关心加密参数的生成过程。

方法

```
from playwright.sync_api import sync_playwright

COUNT = 0
TOKEN = ''


def on_response(response):
    global COUNT, TOKEN
    if '/api/movie/' in response.url and response.status == 200:
        # 获取url中的token参数
        TOKEN = response.url.split('=')[-1]
        # response.json()转换为字典
        mov_list = response.json()
        COUNT = mov_list['count']


def scrape_mov_list():
    global COUNT, TOKEN
    with sync_playwright() as p:
        browser = p.chromium.launch()
        page = browser.new_page()
        # 监听response事件
        page.on('response', on_response)
        # 访问网页
        page.goto('https://spa6.scrape.center/')
        # 等待网页加载完成
        page.wait_for_load_state('networkidle')
        # 访问带有参数的网页
        page.goto(f'https://spa6.scrape.center/api/movie/?limit=
{COUNT}&offset=0&token={TOKEN}')
        # 等待网页加载完成
        page.wait_for_load_state('networkidle')
        # 获取API返回的JSON数据
        mov_list = page.evaluate('() =>
JSON.parse(document.body.innerText)')
        browser.close()

        return mov_list
```

使用示例

```
mov_list = scrape_mov_list()
print(mov_list)
```

运行结果：同上

3.3.5 获取电影详情（Requests 实现）

- 参数：电影 id
- 返回：JSON 格式的电影详情

方法

1. 将电影 id 与密钥 `SECRET` 拼接后进行 base64 编码，得到 `encrypt_id`
2. 根据 path '/api/movie/`encrypt_id`' 获取 `token`
3. 根据加密 id 和 `token` 构造新的 URL
4. 接收新的 URL 的 response，提取 JSON 格式的电影详情
5. 返回电影详情

```
def get_mov_detail(mov_id):
    # 对电影id进行加密
    encrypt_id = base64.b64encode((SECRET + str(mov_id)).encode('utf-8')).decode('utf-8')
    # 构造URL
    url = DETAIL_URL.format(encrypt_id=encrypt_id, token=get_token(args=[f'/api/movie/{encrypt_id}']))
    # 获取电影详情
    return requests.get(url, verify=False).json()
```

使用示例

```
mov_detail = get_mov_detail(3)
print(mov_detail)
```

运行结果：


```
{
  "id": 3,
  "name": "肖申克的救赎",
  "alias": "The Shawshank Redemption",
  "cover": "https://p0.meituan.net/movie/283292171619cdfd5b240c8fd093f1eb255670.jpg@464w_644h_1e_1c",
  "categories": [
    "剧情",
    "犯罪"
  ],
  "regions": [
    "美国"
  ],
  "actors": [
    {
      "...": "..."
    },
    {
      "...": "..."
    }
  ],
  "directors": [
    {
      "...": "..."
    },
    {
      "...": "..."
```

```
    }
  ],
  "score": 9.5,
  "rank": 2,
  "minute": 142,
  "drama": "20世纪40年代末，小有成就的青年银行家安迪（蒂姆·罗宾斯 饰）因涉嫌杀害妻子及她的情人而锒铛入狱……",
  "photos": [

 "https://p1.meituan.net/movie/2aec34359be2d02f87b3b7a5095072ba183155.jpg@106w_106h_1e_1c"
  ],
  "published_at": "1994-09-10",
  "updated_at": "2020-03-07T16:31:54.879934Z"
}
```


3.3.6 获取所有电影详情

- 参数：JSON 格式的电影列表
- 返回：JSON 格式的列表中的所有电影详情

方法

遍历电影列表，调用 `get_mov_detail()`，获取每部电影的详情，最后返回所有电影详情

```python
def get_all_mov_detail(mov_list):
    # 获取所有电影详情
    all_mov_detail = [get_mov_detail(mov['id']) for mov in mov_list['results']]
    # 打印获取电影详情的数量
    print(f'已成功获取 {len(all_mov_detail)} 部电影详情')
    return all_mov_detail
```

使用示例

```python
mov_list = get_mov_list()
all_mov_detail = get_all_mov_detai(mov_list)
print(all_mov_detail)
```

运行结果：


```
已成功获取 101 部电影详情
[
  {"id": 1, "...": "..."},
  {"id": 2, "...": "..."},
  {"id": "...", "...": "..."},
  {"id": 104, "...": "..."}
]
```


4. 总结