Node.js
Secure Coding

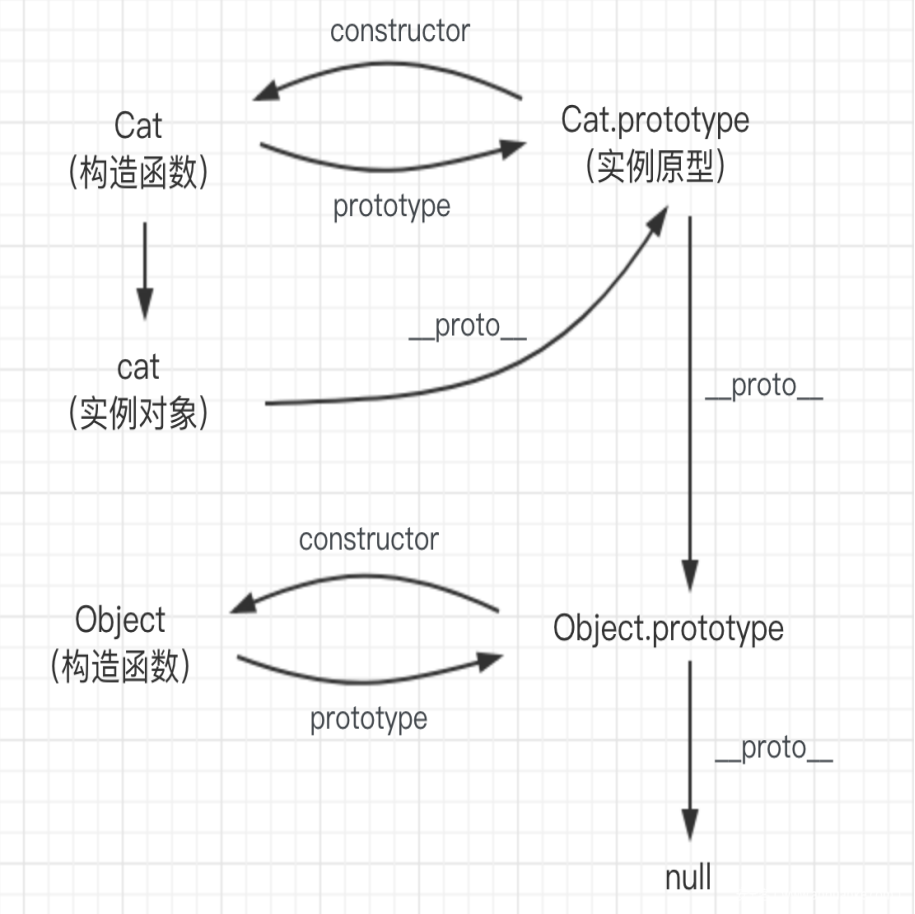
Node.js常见漏洞

- Web常见漏洞:
  - 注入
  - XSS
  - CSRF
  - 越权
  - ...
- Node.js 特色漏洞:
  - 原型链污染
  - 沙箱逃逸
  - 第三方模块 漏洞/恶意代码

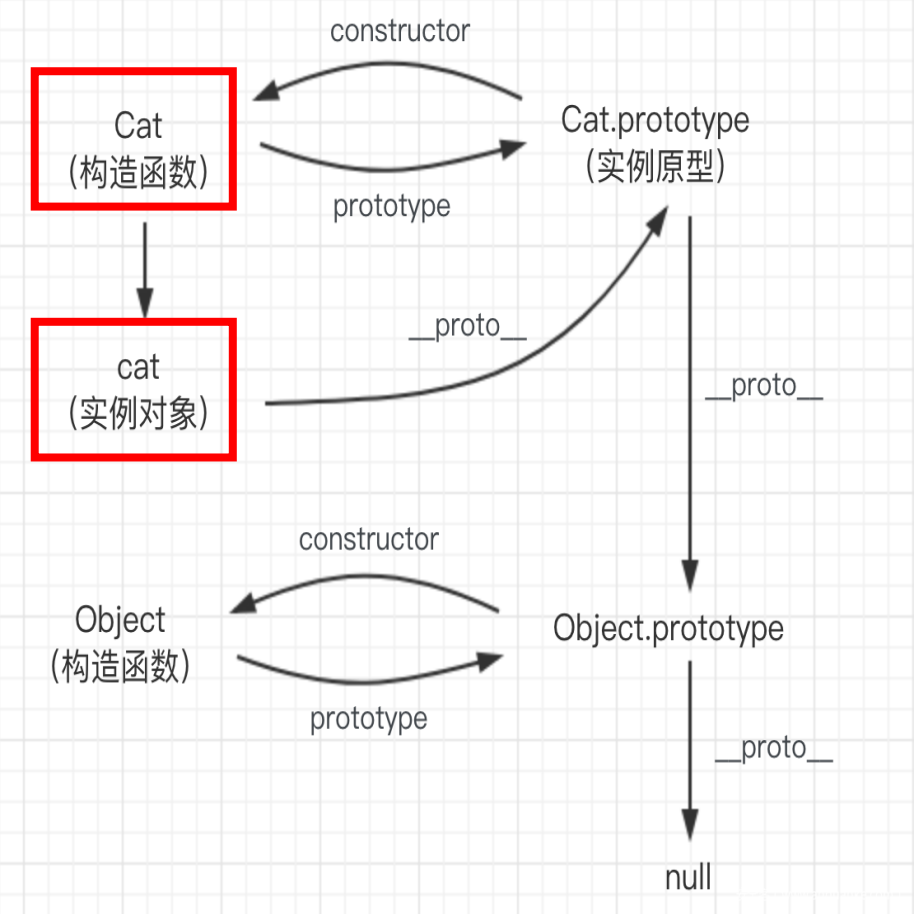
原型链

- 任何对象都是由一个构造函数创建的
- JavaScript中的继承关系：原型链
- 原型：
  - 任何对象都有一个原型对象：
    - 实例对象的内置属性`__proto__` 指向他的原型对象
    - 实例对象的构造函数的prototype属性也指向这个原型对象
    - 每个原型对象的constructor指向实例对象的构造函数
  - 从对象中读取一个缺失的属性时，JavaScript会自动从尝试从原型中获取该属性：原型链继承

原型链

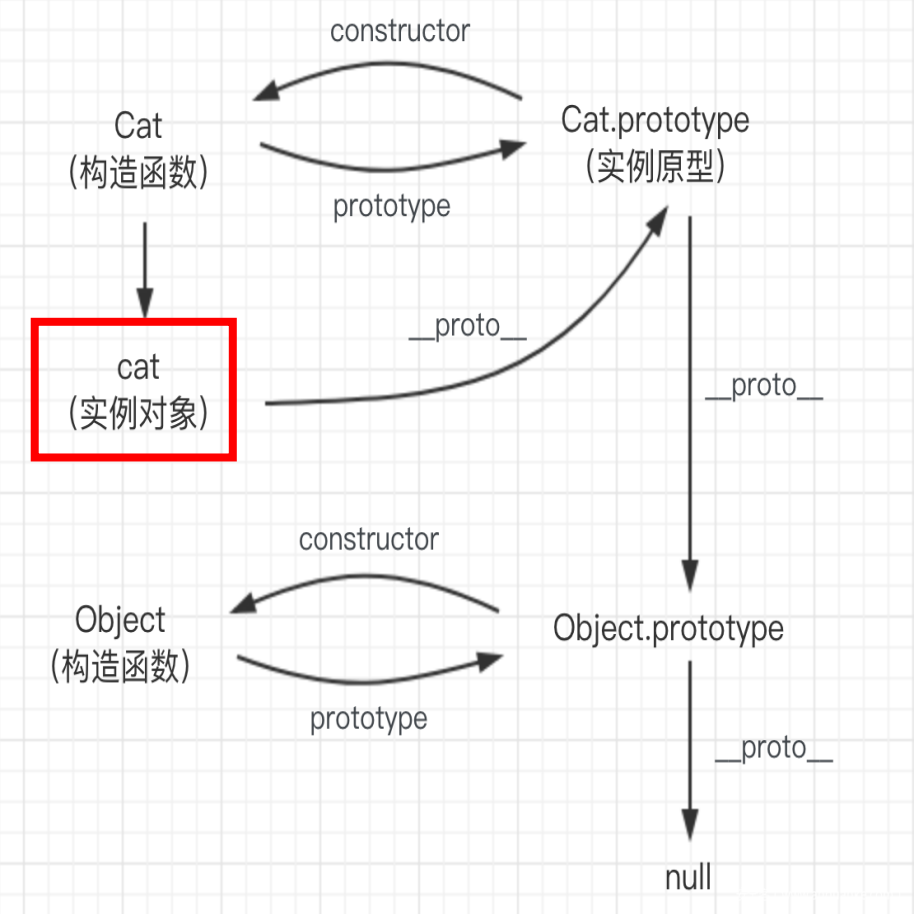

constructor
Cat
(构造函数)
Cat.prototype
(实例原型)
prototype
__proto__
cat
(实例对象)
__proto__
constructor
Object
(构造函数)
Object.prototype
prototype
__proto__
null

原型链



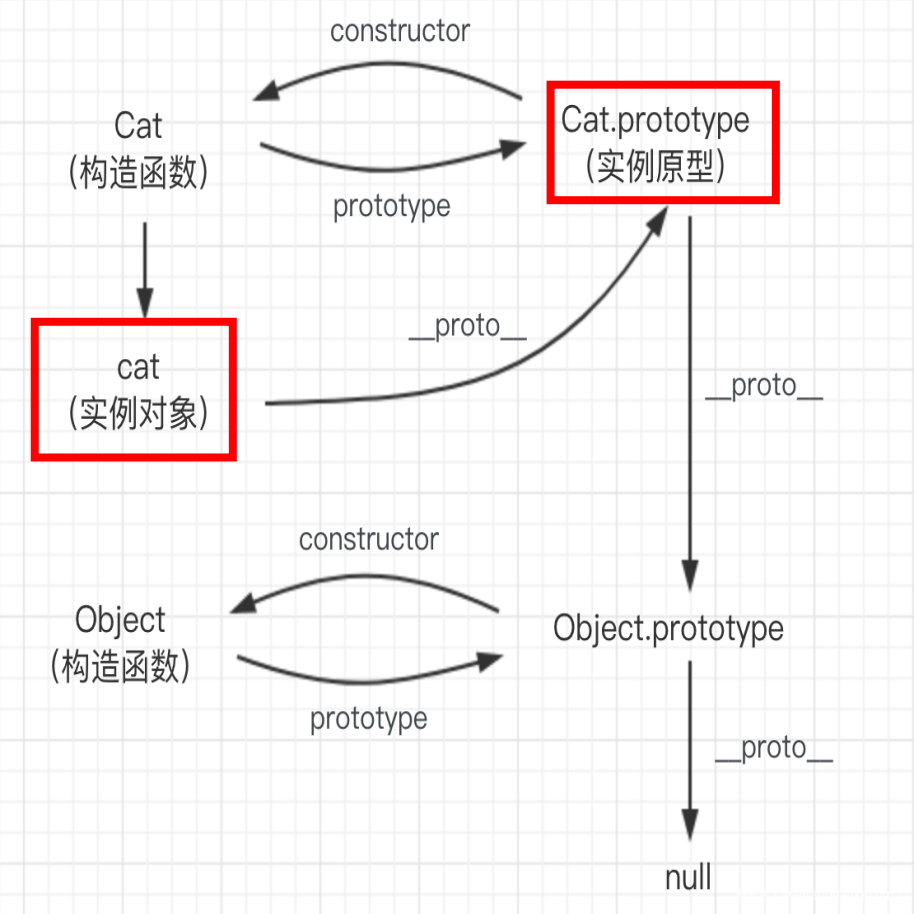

```
> function Cat(){};
undefined
> let cat = new Cat;
undefined
```

原型链



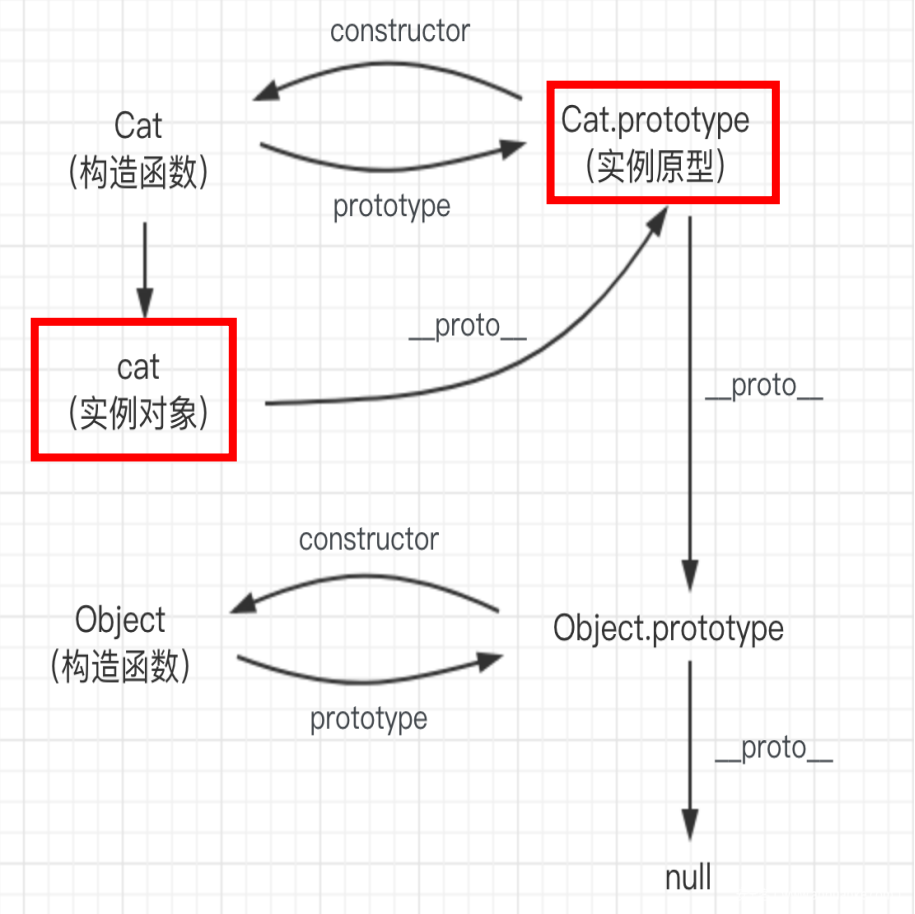

```
> function Cat(){};
undefined
> let cat = new Cat;
undefined
```

原型链




```
> function Cat(){};
undefined
> let cat = new Cat;
undefined
```

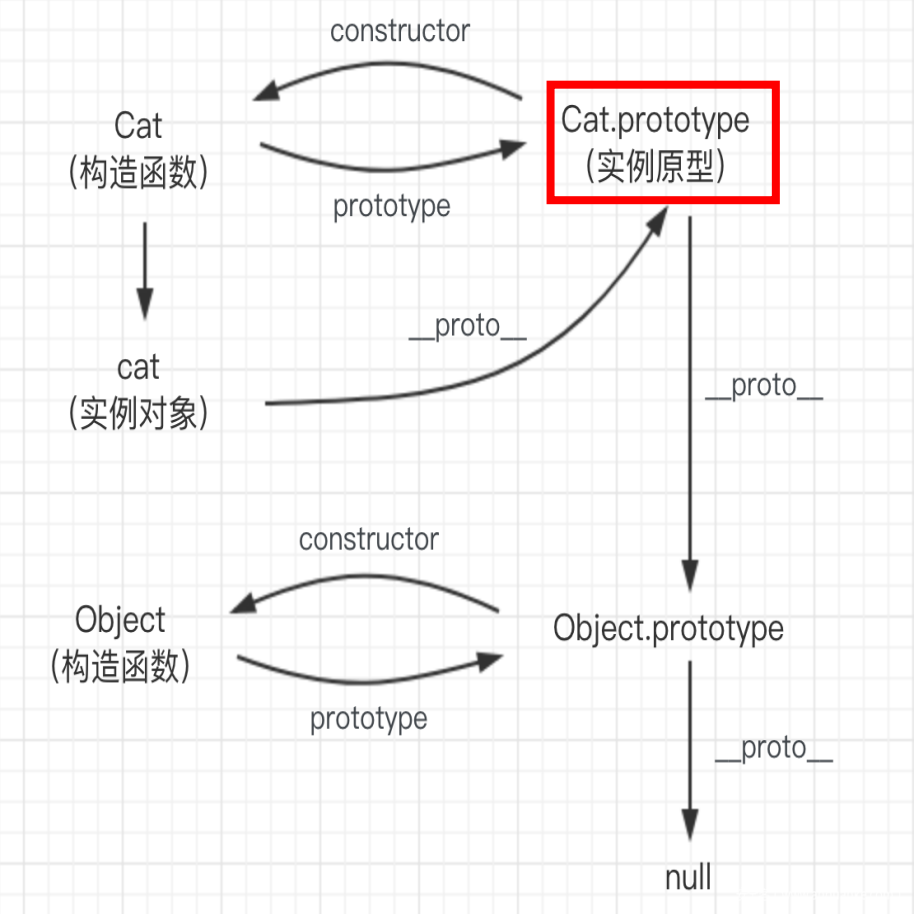
原型链




```
> function Cat(){};
undefined
> let cat = new Cat;
undefined
```

```
> cat.__proto__ == Cat.prototype
true
```

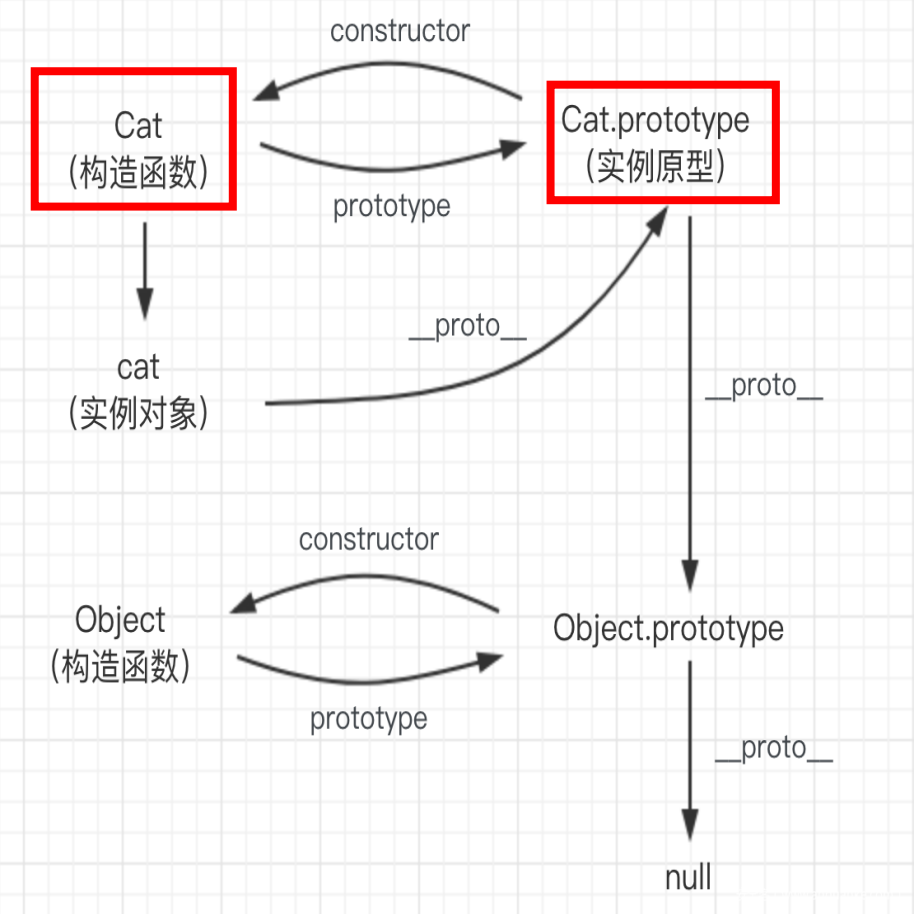
原型链




```
> function Cat(){};
undefined
> let cat = new Cat;
undefined
```

```
> cat.__proto__ == Cat.prototype
true
```

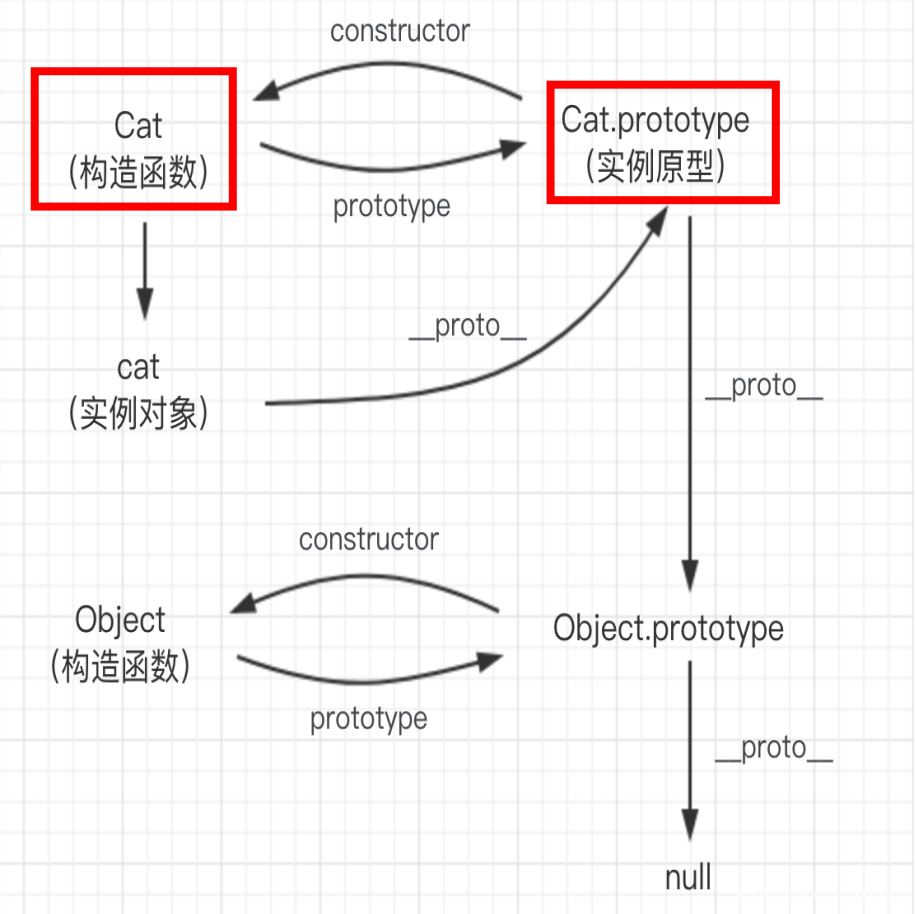
原型链




```
> function Cat(){};
undefined
> let cat = new Cat;
undefined
```

```
> cat.__proto__ == Cat.prototype
true
```

原型链

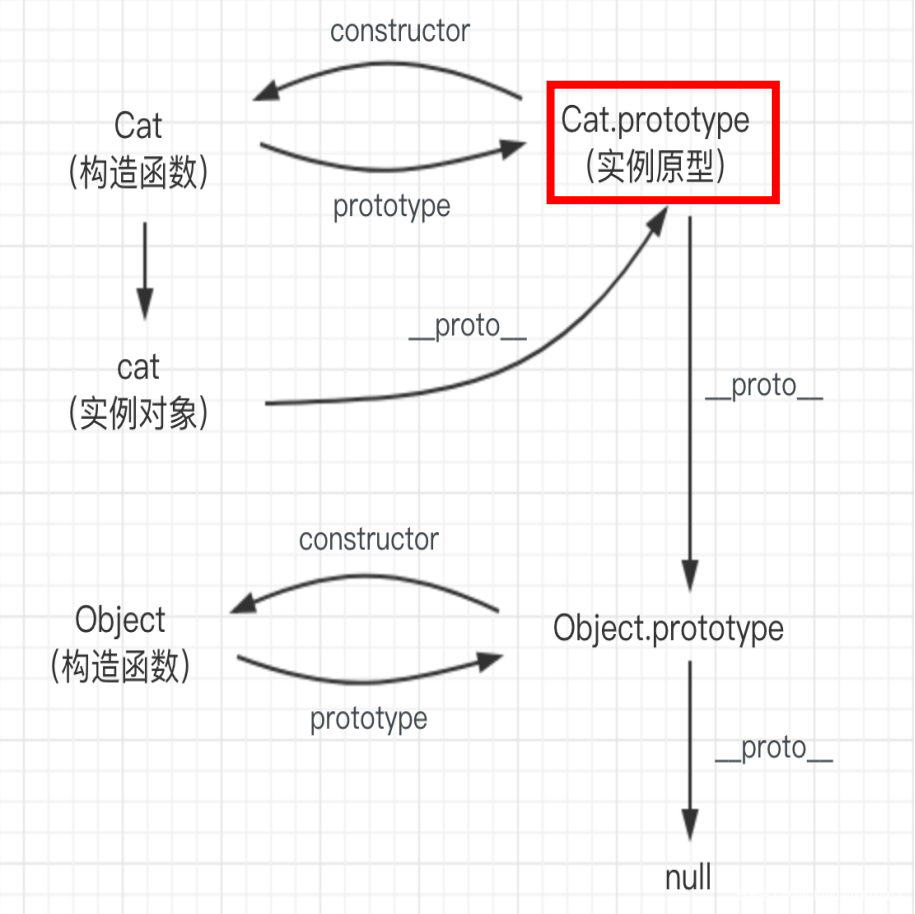



```
> function Cat(){};
undefined
> let cat = new Cat;
undefined
```

```
> cat.__proto__ == Cat.prototype
true
```

```
> Cat.prototype.constructor == Cat
true
```

原型链

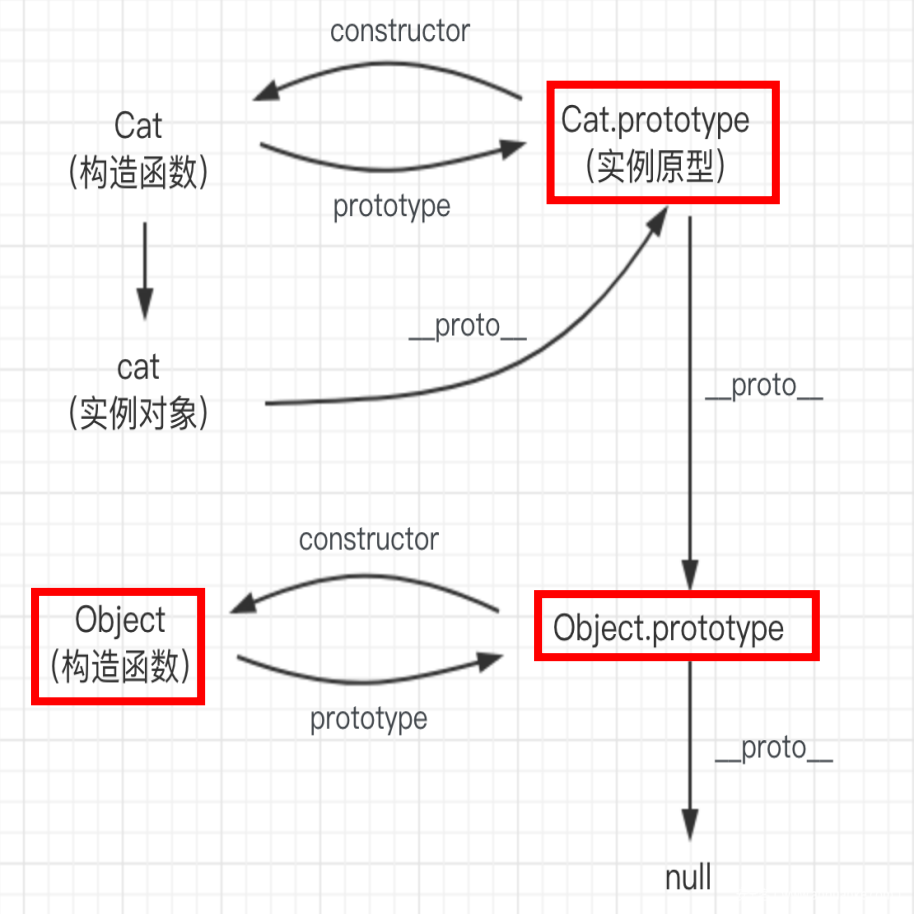



```
> function Cat(){};
undefined
> let cat = new Cat;
undefined
```

```
> cat.__proto__ == Cat.prototype
true
```

```
> Cat.prototype.constructor == Cat
true
```

原型链

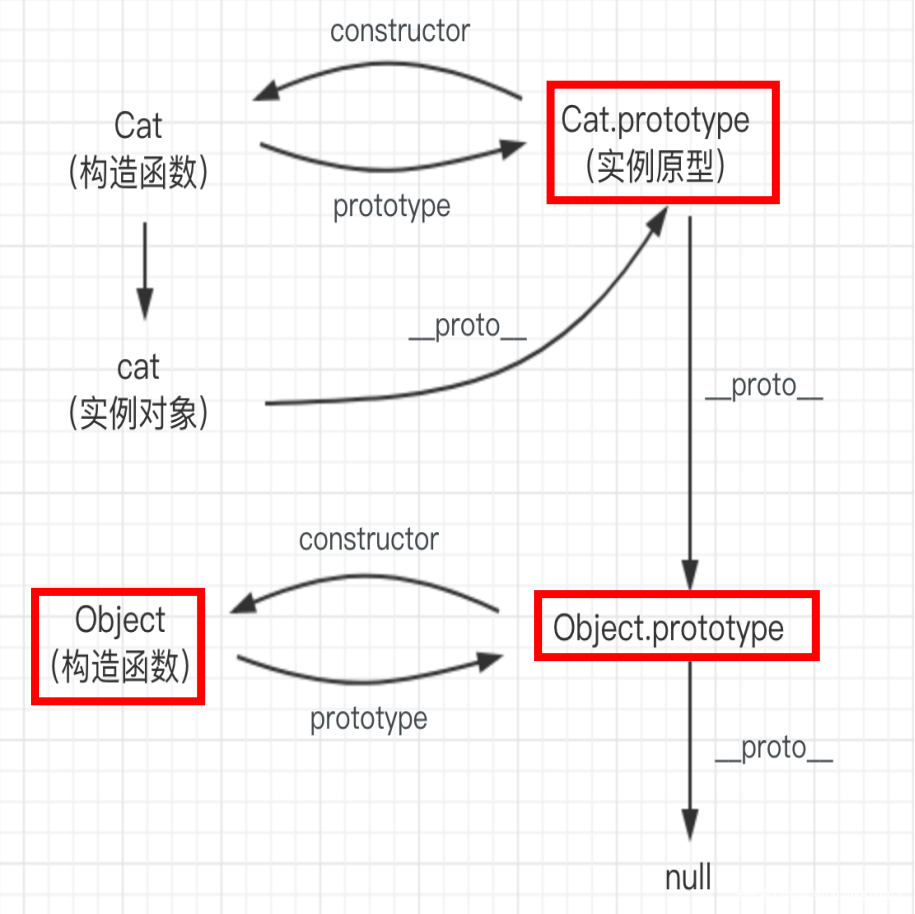



```
> function Cat(){};
undefined
> let cat = new Cat;
undefined
```

```
> cat.__proto__ == Cat.prototype
true
```

```
> Cat.prototype.constructor == Cat
true
```

原型链



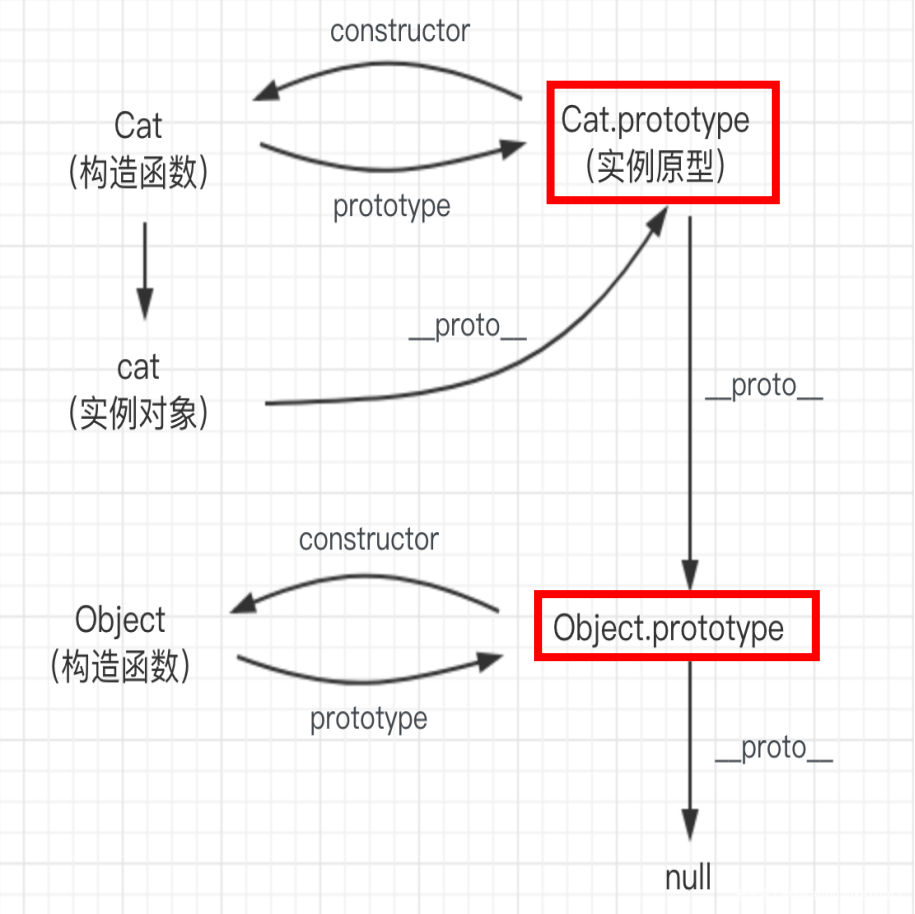

```
> function Cat(){};
undefined
> let cat = new Cat;
undefined
```

```
> cat.__proto__ == Cat.prototype
true
```

```
> Cat.prototype.constructor == Cat
true
```

```
> Cat.prototype.__proto__.constructor == Object
true
```

原型链



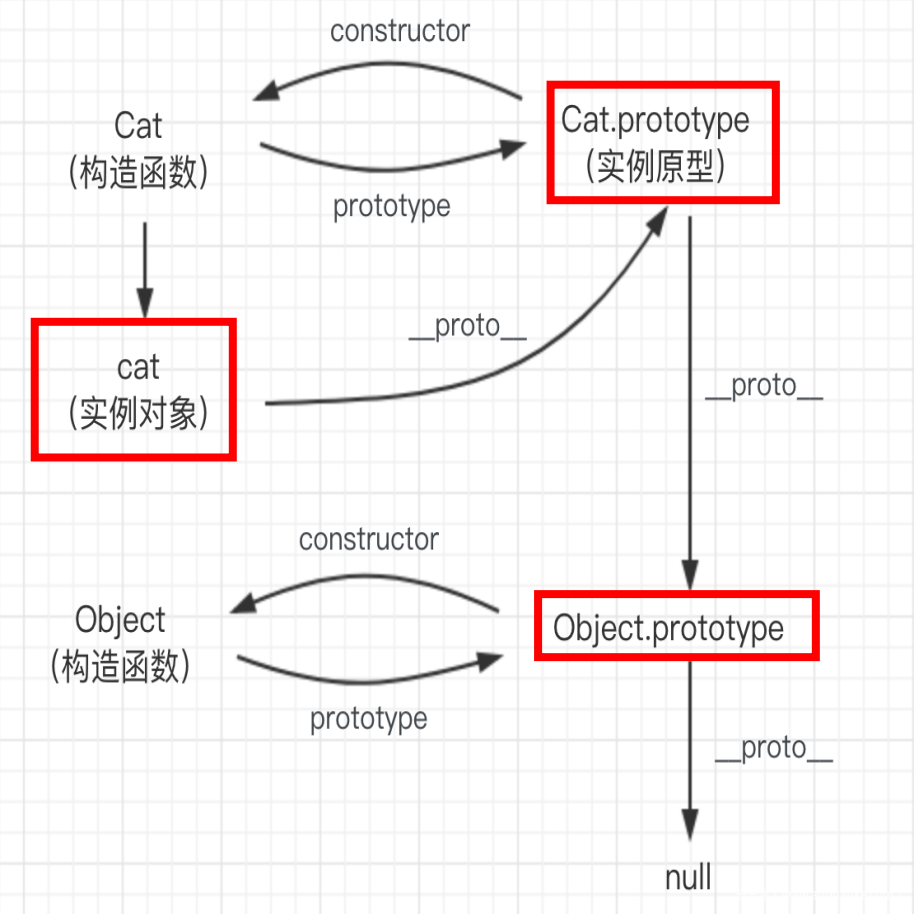

```
> function Cat(){};
undefined
> let cat = new Cat;
undefined
```

```
> cat.__proto__ == Cat.prototype
true
```

```
> Cat.prototype.constructor == Cat
true
```

```
> Cat.prototype.__proto__.constructor == Object
true
```

原型链



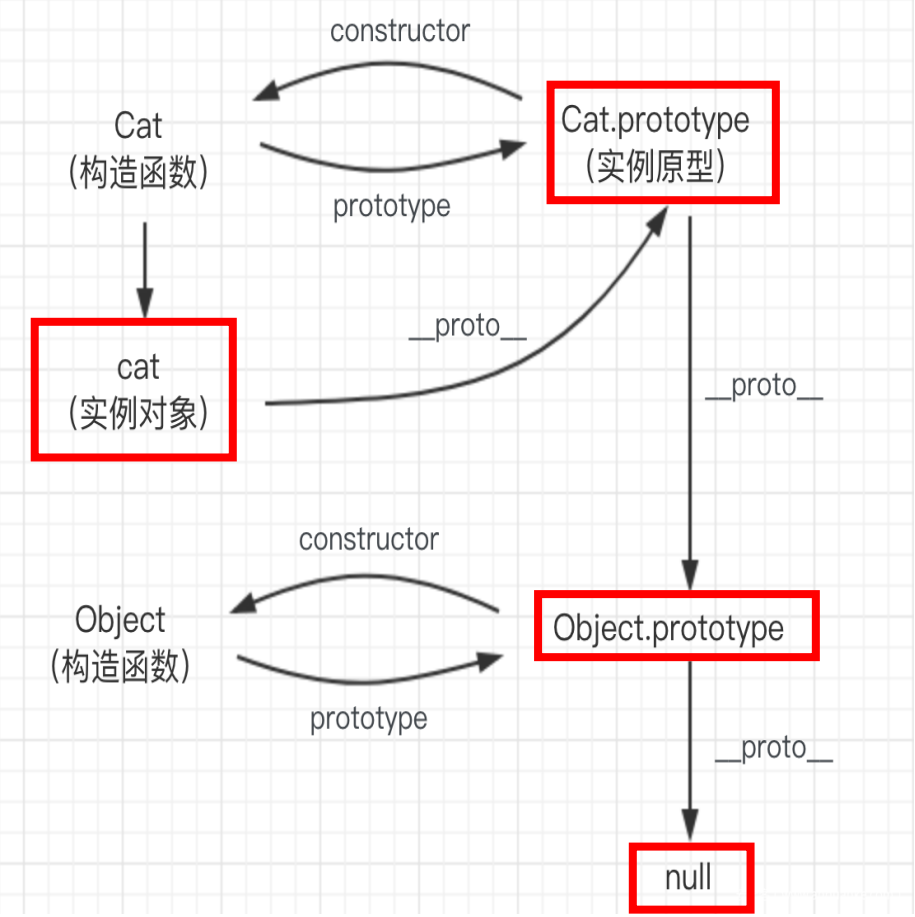

```
> function Cat(){};
undefined
> let cat = new Cat;
undefined
```

```
> cat.__proto__ == Cat.prototype
true
```

```
> Cat.prototype.constructor == Cat
true
```

```
> Cat.prototype.__proto__.constructor == Object
true
```

原型链



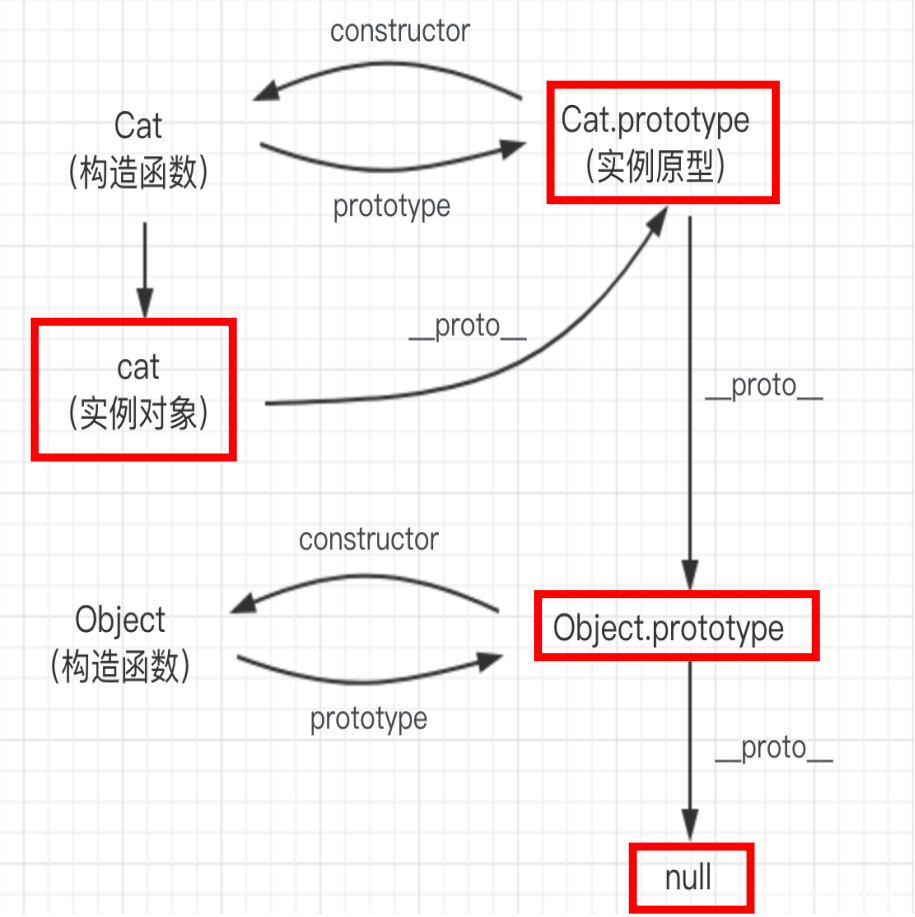

```
> function Cat(){};
undefined
> let cat = new Cat;
undefined
```

```
> cat.__proto__ == Cat.prototype
true
```

```
> Cat.prototype.constructor == Cat
true
```

```
> Cat.prototype.__proto__.constructor == Object
true
```

原型链




```
> function Cat(){};
undefined
> let cat = new Cat;
undefined
```

```
> cat.__proto__ == Cat.prototype
true
```

```
> Cat.prototype.constructor == Cat
true
```

```
> Cat.prototype.__proto__.constructor == Object
true
```

```
> Cat.prototype.__proto__.__proto__
null
```

原型链污染

- 修改其**原型对象**中的属性值
- 使其他通过该构造函数实例化出的对象也具有这个属性的值

```
> // a是一个简单的对象
  a = {}
<· ▶ {}

> // a.__proto__ 是Object的prototype
  a.__proto__ == Object.prototype
<· true

> // 修改a的__proto__（即修改Object的prototype）
  a.__proto__.test = 2
<· 2

> // 创建一个新的Object对象b
  b = new Object
<· ▶ {}

> // Object的prototype已被污染，新增了一个test属性，b也是Object的一个实例
  // 所以b对象也有test属性，通过__proto__向上查找得来
  b.test
<· 2
```

原型链污染的危害

- 拒绝服务
  - 比如：攻击者污染 Object.prototype.toString属性，如果代码库在某个时刻依赖于 someobject.toString()，就会造成拒绝服务
- 远程代码执行
  - 比如：eval(someobject.someattr)，如果攻击者污染 Object.prototype.someattr，那么他们很可能可以利用这一点来执行代码
- 属性注入
  - 如果代码库检查 someuser.isAdmin 的权限，当攻击者污染 Object.prototype.isAdmin 并将其设置为 true 时，他们就能获得管理员权限。

/api/orders

```
app.post('/api/orders', validate(postOrderSchema), (req, res, next) => {

  const clientOrder = _.merge({}, req.body, { ipAddress: req.ip });

  // import clientOrder into database

  res.json({
    status: 'ok',
    ip: clientOrder.ipAddress,
  });
});
```

/api/login

```
function login(username, password) {
  if (username == 'admin' && password == 'thisisaveryveryverylongpassword') {
    return { user: 'admin', admin: true };
  } else {
    return { user: 'guest' };
  }
}

app.post('/api/login', validate(loginSchema), (req, res, next) => {
  user = login(req.body.username, req.body.password);
  if (user.admin) {
    res.json({
      secret: "this is a top secret",
    });
  } else {
    res.json({
      error: "not admin"
    });
  }
});
```

什么情况下会出现原型链污染?

- 对象的递归合并操作
- 对象的克隆操作
- 按路径定义对象属性的操作

对象的递归合并操作

```
function merge(a, b) {
    for (var attr in b) {
        if (isobject(a[attr]) && isobject(b[attr])) {
            merge(a[attr], b[attr]);
        } else {
            a[attr] = b[attr];
        }
    }

    return a;
}
```

```
var a = { "a" : 1, "b" : 2 };
var b = JSON.parse('{"__proto__":{"polluted":1}}');
var c = merge(a, b);
var d = {};
d.polluted // 1
```

曾经受影响的第三方库：

- lodash (CVE-2018-16487)
- hoek (CVE-2020-36604)

对象的克隆操作

- 本质上就是合并操作

```
function clone(a) {
    return merge({}, a);
}

var a  = JSON.parse('{"__proto__":{"polluted":1}}');
var b = clone(a);
var d = {};
d.polluted // 1
```

按路径定义对象属性的操作

- 第三方库实现(如lodash的setWith和set方法)

```
var obj = { b : { "test" : 321 } };
setValue(obj, "b.test", 123);
obj.b.test; // 123
```

```
var obj = {  };
setValue(obj, "__proto__.polluted", 1);
var d = {};
d.polluted // 1
```

如何避免原型链污染?

- 防止更改原型对象：Object.freeze (Object.prototype)
- 对 JSON 输入进行模式验证 (e.g. ajv)
- 使用Map代替Object: let options = new Map();
- 防止对象继承属性: let myObject = Object.create(null);

AMA

沙箱逃逸

- 沙盒是一个隔离的环境，可以在不影响其外部实际代码的情况下安全地执行不受信任的代码。
- vm 模块
  - node:vm模块并非一种安全机制，请勿使用它来运行不受信任的代码！
- vm2 模块
  - 已废弃，因为底层设计原因，漏洞无法修复，请勿使用
- isolated-vm 模块
  - vm2作者推荐
- safeify 模块

/api/calc

```
app.post('/api/calc', validate(mathExp), (req, res, next) => {
  console.log(req.body.exp);
  ans = vm.runInNewContext(req.body.exp);

  res.json({
    ans: ans,
  });
});
```

AMA

第三方模块漏洞

- 如果引入的第三方模块有漏洞，或者有恶意代码，那么将影响到应用本身
- ~~比如前面提到的lodash和hoek的原型链污染漏洞~~

如何防范第三方模块漏洞

- npm audit/npm audit --fix

```
# null @ wl3yvy9k7p in /tmp/nodejs-goof on git:main o [16:44:08]
$ npm audit --json | jq '.vulnerabilities | with_entries(select(.value.severity == "critical" and .value.isDirect == true
))'
{
  "cfenv": {
    "name": "cfenv",
    "severity": "critical",
    "isDirect": true,
    "via": [
      "underscore"
    ],
    "effects": [],
    "range": "<=1.2.3",
    "nodes": [
      "node_modules/cfenv"
    ],
    "fixAvailable": true
  },
  "ejs": {
    "name": "ejs",
    "severity": "critical",
    "isDirect": true,
    "via": [
      {
        "source": 1087812,
        "name": "ejs",
        "dependency": "ejs",
        "title": "High severity vulnerability that affects ejs",
        "url": "https://github.com/advisories/GHSA-6x77-rpqf-j6mw",
        "severity": "high",
        "cwe": [
          "CWE-20"
        ],
        "cvss": {
          "score": 7.5,
          "vectorString": "CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H"
```

恶意的第三方模块

- 供应链攻击
  - 依赖混淆（内部模块和外部模块混用）
    - 内部包发布至 Artifactory 时使用范围(scope)限定
    - 确保在 npmjs.com 上拥有相同的范围
    - package-lock.json的完整性
  - 第三方模块误拼攻击（umbrellajs? unbrellajs?)
    - 审查核实模块名
    - 自动拼写检查工具辅助验证
  - 开发者帐户被接管
    - 2FA
    - 安全培训