代理模式，式如其名——在某些情况下，出于种种考虑/限制，一个对象**不能直接访问**另一个对象，需要一个**第三者**（代理）牵线搭桥从而间接达到访问目的，这样的模式就是代理模式。

代理模式非常好理解，因为你可能天天都在用，只是没有刻意挖掘过它背后的玄机——比如大家耳熟能详的**科学上网**，就是代理模式的典型案例。

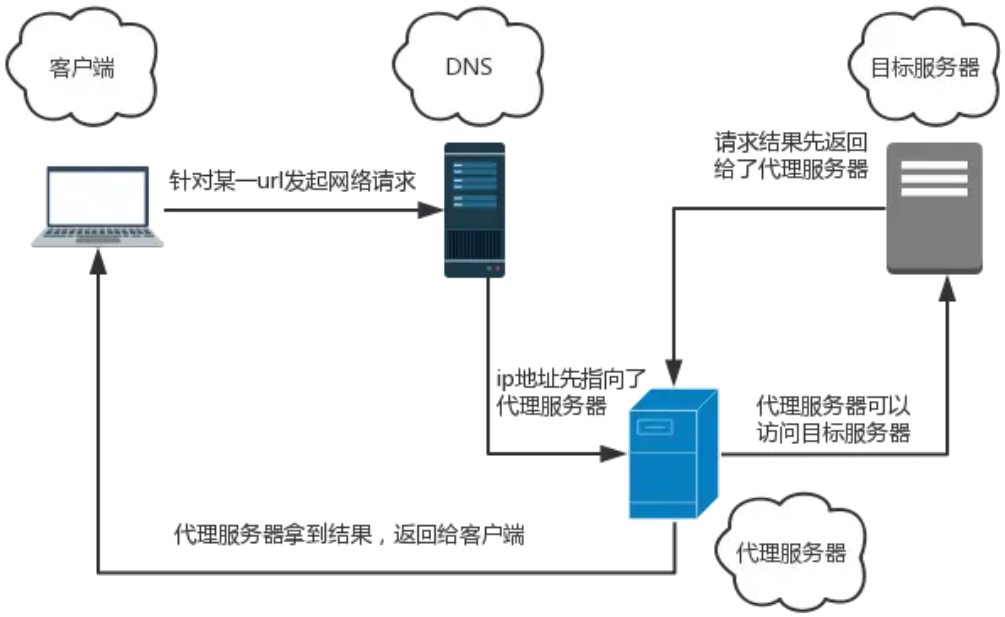
科学上网背后的故事

科学上网，就是咱们常说的 VPN(虚拟专用网络)。大家知道，正常情况下，我们尝试去访问 Google.com，Chrome会给你一个这样的提示：

这是为啥呢？这就要从网络请求的整个流程说起了。一般情况下，当我们访问一个 url 的时候，会发生下图的过程：

为了屏蔽某些网站，一股神秘的东方力量会作用于你的 DNS 解析过程，告诉它：“你不能解析出 xxx.xxx.xxx.xxx（某个特殊ip）的地址”。而我们的 Google.com，不幸地出现在了这串被诅咒的 ip 地址里，于是你的 DNS 会告诉你：“对不起，我查不到”。

但有时候，一部分人为了搞学习，通过访问VPN，是可以间接访问到 Google.com 的。这背后，就是**代理模式**在给力。在使用VPN时，我们的访问过程是这样的：



没错，比起常规的访问过程，多出了一个第三方 —— **代理服务器**。这个第三方的 ip 地址，不在被禁用的那批 ip 地址之列，我们可以顺利访问到这台服务器。而这台服务器的 DNS 解析过程，没有被施加咒语，所以它是可以顺利访问 Google.com 的。代理服务器在请求到 Google.com 后，将响应体转发给你，使你得以间接地访问到目标网址 —— 像这种第三方代替我们访问目标对象的模式，就是代理模式。

婚姻介绍所的故事

我有个同事，技术很强，发型也很强。多年来因为沉迷 coding，耽误了人生大事。迫于寻找另一半的愿望比较急切，该同事同时是多个优质高端婚恋网站的注册VIP。工作之余，他常常给我们分享近期的相亲情感生活进展。

“你们看，这个妹子头像是不是超可爱！”同事哥这天发掘了一个新的婚介所，他举起手机，朝身边几位疯狂挥舞。“哥，那是新垣结衣。。。”同事哥的同桌无奈地摇摇头，没有停下 coding 的手。 同事哥恢复了冷静，叹了口气：“这种婚恋平台的机制就是这么严格，一进来只能看到其它会员的姓名、年龄和自我介绍。要想看到本人的照片或者取得对方的联系方式，得先向平台付费成为 VIP 才行。哎，我又要买个 VIP 了。”

我一听，哇，这婚恋平台把代理模式玩挺 6 啊！大家想想，主体是同事 A，目标对象是新垣结衣头像的未知妹子。同事 A 不能直接与未知妹子进行沟通，只能通过第三方（婚介所）间接获取对方的一些信息，他能够获取到的信息和权限，取决于第三方愿意给他什么——这不就是典型的代理模式吗？

用代理模式开一家婚姻介绍所吧

这样看来，开婚介所确实是个发家致富的好路子。既然暴富的机会就在眼前，那么事不宜迟，我们接下来就一起用 JavaScript 来实现一个小型婚介所。

前置知识： ES6中的Proxy

在 ES6 中，提供了专门以代理角色出现的代理器 —— Proxy。它的基本用法如下：

```
const proxy = new Proxy(obj, handler)
```

第一个参数是我们的目标对象，也就是上文中的“未知妹子”。handler 也是一个对象，用来定义**代理的行为**，相当于上文中的“婚介所”。当我们通过 proxy 去访问目标对象的时候，handler会对我们的行为作一层拦截，我们的每次访问都需要经过 handler 这个第三方。

“婚介所”的实现

未知妹子的个人信息，刚问了下我们已经注册了 VIP 的同事哥，大致如下：

```
// 未知妹子
const girl = {
  // 姓名
  name: '小美',
  // 自我介绍
  aboutMe: '...'（大家自行脑补吧）
  // 年龄
  age: 24,
  // 职业
  career: 'teacher',
  // 假头像
  fakeAvatar: 'xxxx'(新垣结衣的图片地址)
  // 真实头像
  avatar: 'xxxx'(自己的照片地址),
  // 手机号
  phone: 123456,
}
```

婚介所收到了小美的信息，开始营业。大家想，这个姓名、自我介绍、假头像，这些信息大差不差，曝光一下没问题。但是人家妹子的年龄、职业、真实头像、手机号码，是不是属于非常私密的信息了？要想 get 这些信息，平台要考验一下你的诚意了 —— 首先，你是不是已经通过了实名审核？如果通过实名审核，那么你可以查看一些相对私密的信息（年龄、职业）。然后，你是不是 VIP ？只有 VIP 可以查看真实照片和联系方式。满足了这两个判定条件，你才可以顺利访问到别人的全部私人信息，不然，就劝退你提醒你去完成认证和VIP购买再来。

```
// 普通私密信息
const baseInfo = ['age', 'career']
// 最私密信息
const privateInfo = ['avatar', 'phone']

// 用户（同事A）对象实例
const user = {
    ...(一些必要的个人信息)
    isValidated: true,
    isVIP: false,
}
```

```
// 掘金婚介所登场了
const JuejinLovers = new Proxy(girl, {
  get: function(girl, key) {
      if(baseInfo.indexOf(key)!==-1 && !user.isValidated) {
          alert('您还没有完成验证哦')
          return
      }

      //...(此处省略其它有的没的各种校验逻辑)

      // 此处我们认为只有验证过的用户才可以购买VIP
      if(user.isValidated && privateInfo.indexOf(key) && !user.isVIP) {
          alert('只有VIP才可以查看该信息哦')
          return
      }
  }
})
```

以上主要是 getter 层面的拦截。假设我们还允许会员间互送礼物，每个会员可以告知婚介所自己愿意接受的礼物的价格下限，我们还可以作 setter 层面的拦截。：

```
// 规定礼物的数据结构由type和value组成
const present = {
    type: '巧克力',
    value: 60,
}

// 为用户增开presents字段存储礼物
const girl = {
  // 姓名
  name: '小美',
  // 自我介绍
  aboutMe: '...'（大家自行脑补吧）
  // 年龄
  age: 24,
  // 职业
  career: 'teacher',
  // 假头像
  fakeAvatar: 'xxxx'(新垣结衣的图片地址）
  // 真实头像
  avatar: 'xxxx'(自己的照片地址),
  // 手机号
  phone: 123456,
  // 礼物数组
  presents: [],
  // 拒收50块以下的礼物
  bottomValue: 50,
  // 记录最近一次收到的礼物
  lastPresent: present,
}

// 掘金婚介所推出了小礼物功能
const JuejinLovers = new Proxy(girl, {
  get: function(girl, key) {
    if(baseInfo.indexOf(key)!==-1 && !user.isValidated) {
        alert('您还没有完成验证哦')
        return
    }
```

```
    //...(此处省略其它有的没的各种校验逻辑)

    // 此处我们认为只有验证过的用户才可以购买VIP
    if(user.isValidated && privateInfo.indexOf(key) && !user.isVIP) {
        alert('只有VIP才可以查看该信息哦')
        return
    }
  }

  set: function(girl, key, val) {

    // 最近一次送来的礼物会尝试赋值给lastPresent字段
    if(key === 'lastPresent') {
      if(val.value < girl.bottomValue) {
          alert('sorry，您的礼物被拒收了')
          return
      }

      // 如果没有拒收，则赋值成功，同时并入presents数组
      girl[lastPresent] = val
      girl[presents] = [...presents, val]
    }
  }

})
```

看来婚介所这条路，真是不太好走。掌握了代理模式的常见使用方式之余，我们也敬这位同事哥是条汉子，希望他早日脱离苦海~

不过如果认为代理模式的本领仅仅是开个婚介所这么简单，那就太小瞧它了。代理模式在前端领域一直是一种应用十分广泛的设计模式，在下个小节，我们将会选取其中最典型、最实用的四种类型的应用实践，帮助大家掌握代理模式在业务开发中的应用场景和使用方法。