1. **验证码生成原理：**

验证码的功能一般是防止使用程序恶意注册、暴力破解或批量发帖而设置的。所谓验证码，就是将一串随机产生的数字或符号，生成一幅图片，图片里加上一些干扰象素（防止OCR），由用户肉眼识别其中的验证码信息，输入表单提交网站验证，验证成功后才能使用某项功能

**1.1图片验证码的原理：**

一般验证码的生成方法都是相同的,主要的步骤都有两步

第一步是：随机出一系统验证码的数字或字母，可以任意设定生成验证码的位数，顺便把随机生成的数字或字母写入Cookies 或者 Session.

第二步是：用第一步随机出来的数字或字母来合成图片，数字和字符可以通过特定的几何变换和旋转，变形成随机字符或者数字。先生成验证码图片，接着生成随机生成器，清空图片背景色，由自己对背景颜色进行填充，然后就要添加噪声，噪声可以分为画图片的背景噪音线，画图片的前景噪音点，画图片的边框线。

所以验证码的复杂度主要是第二步来完成，可以根据自己所要的复杂度来设定。根据验证码的生成位置可以分成前段生成和后台生成。前端通过js进行生成一组随机数字或者字符；后台通过java或者php将一组随机数字或者随机字符，绘制到图片中，同时加上点噪点。最后将图片送到前端显示

针对图片验证码，市面上现有的类型如下几种：

（1）四位数字，随机的一数字字符串，最原始的验证码，验证作用几乎为零。

（2）随机数字图片验证码。图片上的字符比较中规中矩，有的可能加入一些随机干扰素，还有一些是随机字符颜色，验证作用比上一个好。没有基本图形图像学知识的人，不可破！

（3）各种图片格式的随机数字+随机大写英文字母+随机干扰像素+随机位置。

（4）汉字是注册目前最新的验证码，随机生成，打起来更难了,影响用户体验，所以，一般应用的比较少

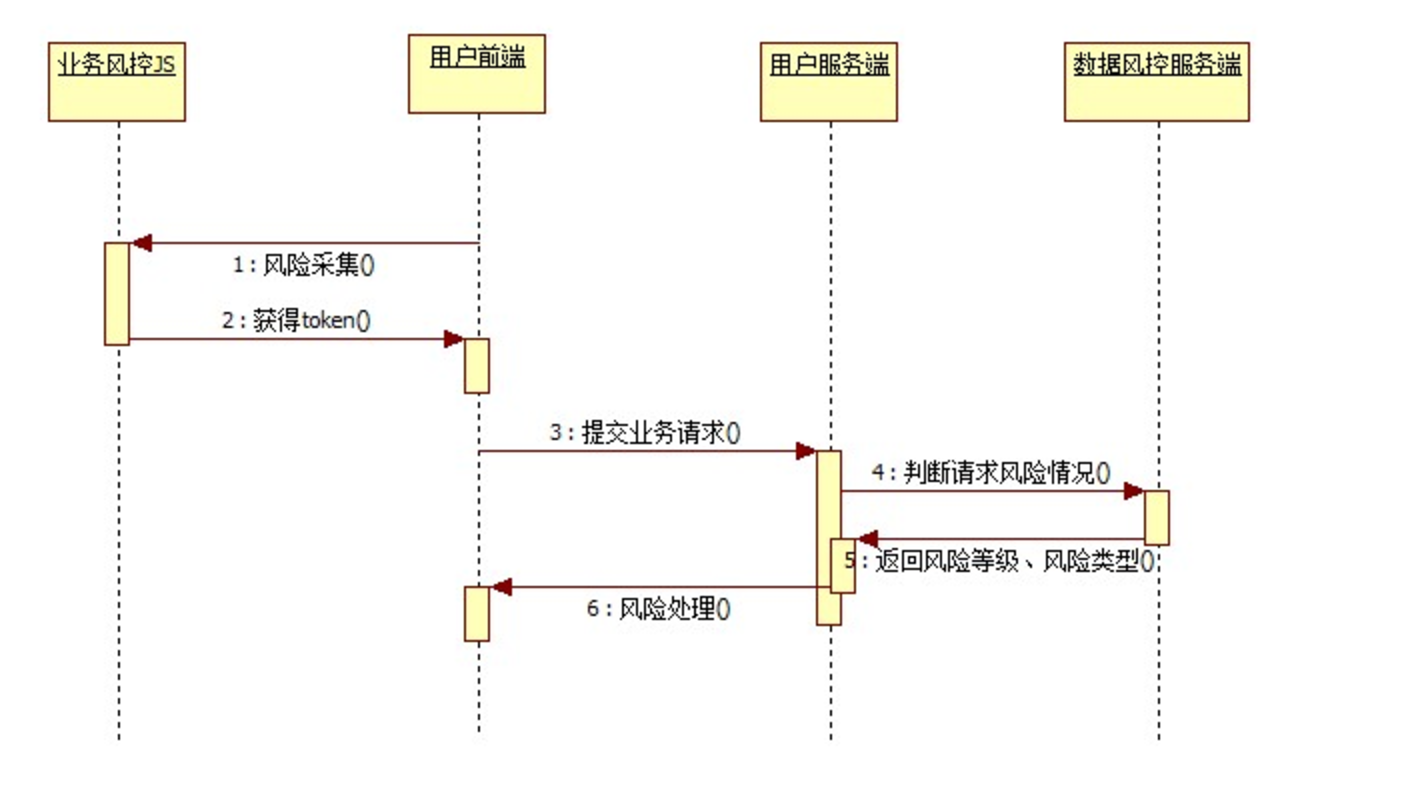
对于识别验证码的程序来说，最难得部分是验证字符的切割和特征码的建立，而国内相关验证码生成技术重点事在验证码加很多干扰素，干扰线，影响效果不说，还达不到很好的效果；所以，要想使自己验证码难于本识别，只做下面两点就够了。

（1）字符粘连，最好所有的字符都有粘连的部分；

（2）不要使用规格字符，验证码的各个部分使用不同比例的缩放或者旋转。

**1.2滑块拖动验证码的原理：**

滑动验证码是根据人在滑动滑块的响应时间，拖拽速度，时间，位置，轨迹，重试次数等来评估风险。其系统间交互流程如下图所示：



响应时间，拖拽速度，时间，位置，轨迹，重试次数等。这些因素能够构成一个采样结果或者辨识特性。根据这些特性可以猜测，你是人或者其他东西。或者能够猜测，你是谁或者什么东西。猜测结果中会涉及到另一个问题，那就是概率，比如80%的概率是人，那么就会放行，30%的概率是人。

原理有两部分组成，一个是分割图片，一个是将图片组合，如下两点：

（1）将完整的图片扣除一部分，变成了两块残缺的图片。然后通过用户用鼠标拖动滑块来进行拼图，拼图正确则完成验证

（2）对鼠标拖动过程会有坐标记录，如果是机器自动拖动，则会验证失败。人工手动拖动，才符合人工特点，才能验证成功。只要鼠标发生移动就记录xy坐标和时间，最后对这个序列以及最终停下的位置分别加密，发给服务器判定。

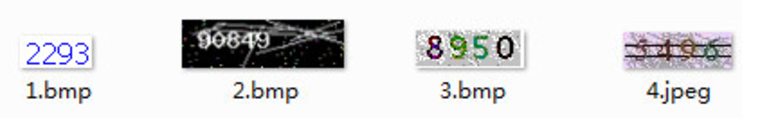
**1.3点触验证码的原理：**

（1）网站给出一组不同种类的图片，根据要求选中特定种类的的图片，将点击结果提交到后台，如果选择的种类符合要求，则通过验证。

（2）网站前端出给一张带有多个文字或者多个图像的图片，根据要求按特定的顺序选中若干个文字或者若干个图像，将点击结果提交到后台，如果选择的种类符合要求，则通过验证。

1. **验证码破解原理：**

**2.1图片验证码的破解原理：**



对照1.1中的四种图片验证码类型，有以下几种破解思路：

第一种，最容易，图片背景和数字都使用相同的颜色，字符规整，字符位置统一。

第二种，看似不容易，其实仔细研究会发现其规则，背景色和干扰素无论怎么变化，验证字符字符规整，颜色相同，所以排除干扰素非常容易，只要是非字符色素全部排除即可。

第三种，看似更复杂，处理上面提到背景色和干扰素一直变化外，验证字符的颜色也在变化，并且各个字符的颜色也各不相同。

第四种，除了第三个图片上提到的特征外，又在文字上加了两条直线干扰率，看似困难其实，很容易去掉

验证码识别一般分为以下几个步骤：

（1）取出字模 识别验证码，毕竟不是专业的OCR识别，并且，由于各个网站的验证码各不相同，所以，最常见的方法就是就是建立这个验证码的特征码库。去字模时，我们需要多下载几张图片，使这些图片中，包括所有的字符，我们这里的字母只有图片，所以只要收集到包括0-9的图片即可。

（2）二值化 二值化就是把图片上的验证数字上每个象素用一种数字表示1，其他部分用0表示。这样就可以计算出每个数字字模，记录下这些字模来，当作key即可。

（3）计算特征 把要识别的图片，进行二值化，得到图片特征。

（4）对照样本 把步骤3种的图片特征码和验证码的字模进行对比，得到验证图片上的数字。

针对如何取出干扰素，首先要清楚干扰素的一个重要特征是，不能影响验证码的显示效果，所以制作干扰素时它的RGB可能低于或者高于某个特定值，比如某验证码图片，干扰素的RGB各项值是不会超过145的，所以，这样我们就很容易去掉干扰素了。

下面举例进行说明：



第一步：二值化。把验证码的部分用 1 表示，背景部分用 0 表示出来，识别方法很简单，我们打印出验证码整张图片的 RGB ，然后分析其规律即可，通过 RGB 码，我们很容易分辨出上面这张图片的 R 值大于 120 ， G 和 B 的值小于 80 ，所以依据这个规则我们很容易把上面的图片二值化。

再来看看上面的第三种验证码图片



刚看上去，感觉很复杂。验证码的图片每次背景色都不相同，且不是单色，各个验证码数字的颜色每次也各不相同。貌似很难二值化，其实我们打印出其 RGB 值很容易就发现。无论验证数字颜色如何变化，该数字的 RGB 值总有一个值小于 125 ，所以通过如下判断 $rgbarray['red'] < 125 || $rgbarray['green']<125|| $rgbarray['blue'] < 125 我们就很容易分辨出哪里是数字，哪里是背景。

我们能够找到这些规律的原因是，在制作验证码的干扰素时，为了使干扰素不影响数字的显示效果，必须使用干扰素的 RGB 和数字 RGB 相互独立，互不干扰。只要懂得这个规律，我们就很容易实现二值化。

我们找到的 120 ， 80 ， 125 等阈值，可能和实际的 RGB 有出入，所以，有时二值化后，会有部分地方出现 1 ，对于验证码上固定位置显示数字，这种干扰没有太大意义。但是对于验证码位置不确定的图片来说，在我们切割字符时，很可能造成干扰。所以，在二值化后要进行去噪处理。

第二部：去噪处理。去噪的原理很简单，就是把孤立的有效的值去掉，如果噪点比较高，要求的效率也比较高的话，这里面也有很多工作要做。幸好这里我们不要求这么高深，我们使用最简单的方法就可以，如果一个点为 1 则判断这个点的上下左右上左上右下左下右 8 个方位上数字是否为 1 ，如果不为 1 ，就认为是一个燥点，直接设置为 1 即可。

第三部：切割字符。切割字符的方法有很多种，这里采用最简单的一种，先垂直方向切割成为字符，然后在水平方向去掉多于的 0000.

按上面的方法会把 dw 字符切割成一个字符，这是错误的切割，所以这里我们涉及到粘连字符的切割。

第四步：粘连字符切割。制作验证码时，规则字符的粘连很容易分割开，如果字符本身有缩放，变形就很难处理，经过分析，我们可以发现，上面的字符粘连属于很简单的方式，只是规则字符的粘连，所以处理这种情况，我们也使用很简单的处理方式。当完成分割操作后，我们不能马上确定分割的部分就为一个字符，要进行验证，验证的关键因素就是，切割下来的字符的宽是否大于阈值，这个阈值的取舍标准是，一个字符无论怎么旋转变形都不会大于这个阈值，所以，如果我们切割的块大于这个阈值，就可以认为这是一个粘连字符；如果大于两个阈值之和，就认为是三个字符粘连，以此类推。知道这个规则后，切割粘连字符也就很简单了。如果我们发现是粘连字符块，直接平分这个块为两个或者多个新的块就可以。当然为了更好的还原字符，我一般都采用平分 +1 ， -1 对字符块的部分进行适当的补充。

第五步：匹配字符。对于旋转字符的特征码建立，有很多种方法，这里就不做深入研究了。我这里使用的最简单的方式，为所有字符的所有情况建立匹配库，所以在我提供的代码种增加了 study 操作，其目的就是，先有人手工识别图片的验证码，然后通过 study 方法，写入特征码库。这样写入的图片数据越多，验证识别的准确行也就越高。

经过以上步骤，我们基本上可以识别现在互联网上大部分的验证码，这里我们都是使用的最简单的方法，没有使用任何 OCR 知识。

**2.2滑块拖动验证码的破解原理：**

要破解滑块拖动验证码首先要解决下面几个问题：

（1）最终停止位置xpos

每次验证请求都会得到三张图，fullbg：完整的验证图形、slice：拼图那一小块、bg：拼图对应那一块加阴影的fullbg，对比fullbg和bg，哪一列像素上开始发生较大变化那就是xpos

如果以后不提供fullbg，只要得到几张不同的bg，求它们的交集即可得到fullbg，不需要机器学习算法

（2）鼠标移动轨迹a

根据xpos可以确定大概需要生成多少帧

x随帧编号产生不均匀增加，直至xpos，开始是用arctan函数模拟的，后来发现完全没必要，建个bitmap[xpos]，从里面随机挑选不重复项选就行了

y用长周期小振幅的sin函数模拟

t随时间产生不均匀增加，并加入一些跳变

用以上方法填充a

（3）加密

不需要破解，从js中剥离加密相关的函数，放入js虚拟机，传入xpos和a并执行，获得加密后结果

**滑块拖动验证码的破解方法：**

**（1）初步利用js、canvas破解**

读取原图： 读两张图片，用 PS 补成原图。

对比图片：利用canvas的特性 getImageData 读取像素，然后和你想要识别的图片对比，找出被挖出去的部分。

计算补全： 然后就简单啦，计算好被挖出部分在x 和 y 方向的偏移量，然后把 小图移动过去。

模拟：页面模拟：除了知道怎么识别图片位置，还需要知道怎么在页面模拟鼠标拖动操作，将图片移动到合适的位置。

在页面拖动一个元素，差不多看成是3个事件的组合：mousedown，mousemove，mouseup。

最后就是用js代码模拟拖动。

（2）综合方法进行破解

第一步：模拟浏览器

可以selenium＋phantomjs的组合， selenium是是一个自动化测试工具，模拟浏览器操作，可以实现鼠标的操作，phantomjs可是一个无界面浏览器。在写破解程序之前，把需要预先填好的信息都提前完成了，让对面服务器认为这就是一个普通的用户。

第二步：找到拼图的位置

首先先通过selenium+phantomjs的截图的功能获得原始图片，再将鼠标点击住滑块（clickandhold），等待几秒，截图获得拼图的图片。通过比较像素点的方法找到缺失位置的的，当像素点差异过了一定范围，这时在比较该点附近的一些点来比较，确定是不是拼图的边界。附近的比较点需要深入研究，横向比较（x轴）和竖向比较（y轴）考虑的都不太全面，要考虑拼图上的凸起和凹陷的情况

之所以不能直接通过差异来确定拼图缺失块边界呢，原因是对比的两张图是截图得来的，或多或少会有一些差异，很可能就有一两个点被程序认为是目标位置的边界。同时也有可能对拼图的阴影位置误判。

第三步：滑动轨迹

前面几步我们得到滑块应该滑到的位置，当我们让程序滑动时会出现下面的情况。

我们将滑块滑到了正确的位置，却被告知验证错误，其实是被怀疑是机器人了。应该是人机特征。解决思路是：先收集了自己的滑动数据三十组，之后用数据做了个线性回归，最后可以通过。

回归算法的缺点是需要严格的假设的，在做这个之前先抽取几组数据做 时间－位移的折线图，发现速度轨迹其实是先快，接近目标点后开始慢的。可以把函数近似看成两个一元一次函数的组合，分别作回归。

**2.3点触验证码的破解原理：**

（1）针对1.3中的（1）的破解方法：

目前已有的破解思路是通过程序用OCR光学字符识别出图片内容，并给出名称描述，每个名称和图片一一对应。根据给出的文字描述，按验证要求模拟点击图片，从而完成验证。

用OCR光学字符识别出图片内容需要自己有大量的图片资源，并机器学习算法对这些图片进行分类训练，个人开发者很难完成这么大的工作。目前可以采用百度图片的API，辅助进行图片内容种类识别。但就当前的效果而言，正确率较低。