密码算法详解——AES

ReadingLover 2015-03-13 原文

0 AES简介

1997年1月2号，美国国家标准技术研究所宣布希望征集一个安全性能更高的加密算法（AES）[3]，用以取代DES。我们知道DES的密钥长度是64 bits，但实际加解密中使用的有效长度只有56 bits，因此算法的理论安全性是2^{56}。但随着制造工艺的不断进步，计算机的计算能力也越来越强，DES将不能提供足够的安全性。AES得到了全世界很多密码工作者的响应，先后有很多人提交了自己的设计方案。截止到最后一轮评选，仍然有5个候选算法：Rijndael，Serpent，Twofish，RC6和MARS。最终经过严格的性能测评，Rijndael算法获胜，因此AES算法也叫Rijndael。

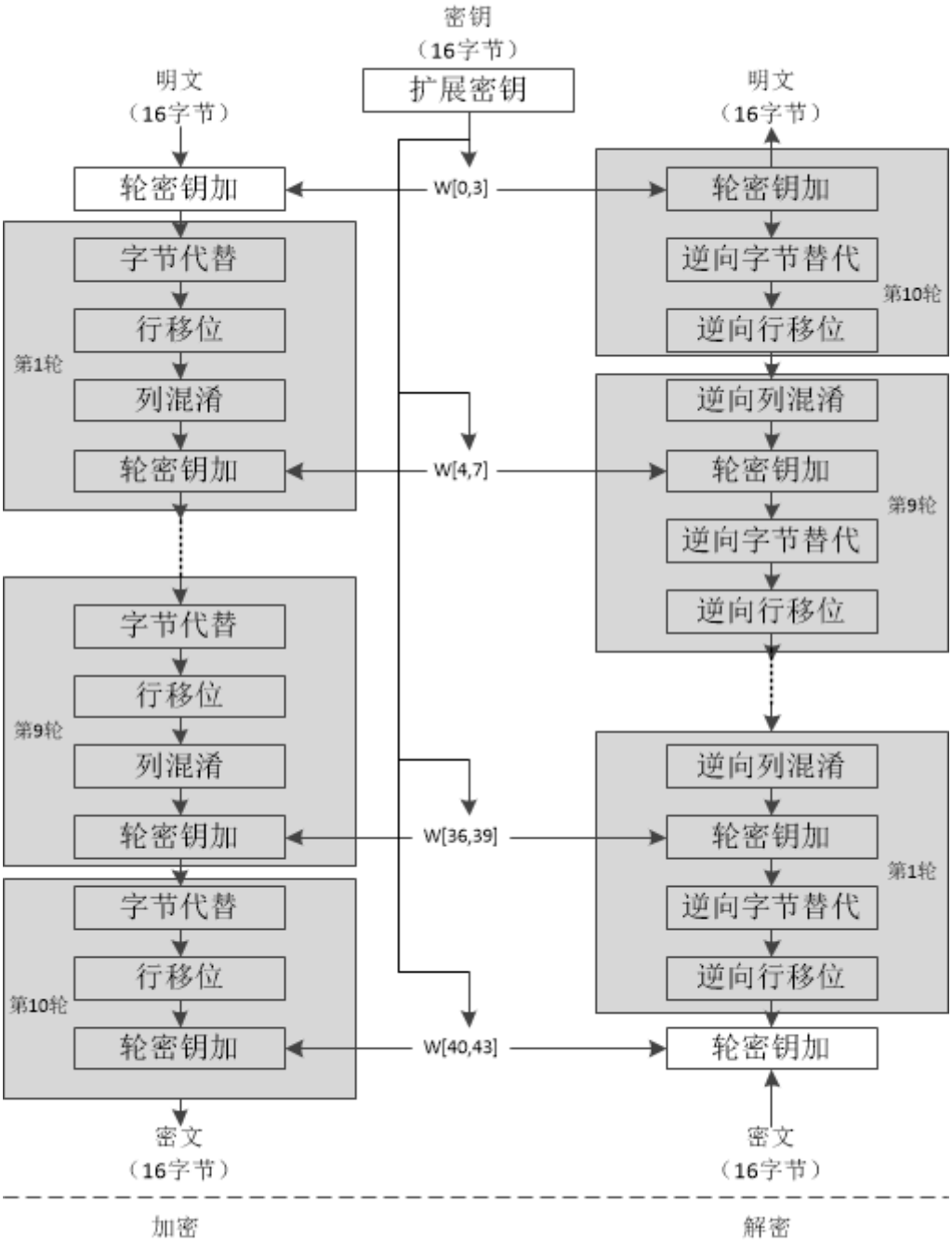
AES由Joan Daemen和Vincent Rijmen设计。根据算法密钥的长度，AES有3种不同方案用以满足不同的场景需求，分别是AES-128、AES-192和AES-256。本文主要对AES-128进行介绍，另外两种的思路基本一样，只是密钥扩展的过程会稍有不同，加解密的轮数会适当增加，但加解密的操作都是一样的。

接下来让我们一起探讨AES内部的结构。

（本文只是对AES算法的各个部件、基本原理进行简单介绍，旨在加深对算法流程的了解。在正式软件运用中也不推荐自己编写代码，很多开源项目都有实现；如果是为了加深对算法实现细节的了解，写写也挺好的。AES算法的部件在对称密码领域特别是分组密码领域常有使用，但由于数学知识的缺陷，S盒以及列混淆中使用的矩阵的很多性质没有进行介绍，读者有兴趣可以自行参考相关文献。）

1 算法流程

AES加解密的流程图如下：



AES加密过程涉及到4种操作：字节替代（SubBytes）、行移位（ShiftRows）、列混淆（MixColumns）和轮密钥加（AddRoundKey）。从上图可以看出：1）解密过程的每一步分别对应操作的逆操作，2）加解密所有操作的顺序正好是相反的。正是由于这两点保证了解密能够正确地恢复明文。加解密中每轮的密钥分别由初始密钥扩展得到。算法中16字节的明文、密文和轮密钥都以一个4x4的矩阵表示。

接下来分别对上述5种操作进行介绍。

1.1 字节代替

字节代替的主要功能是通过S盒完成一个字节到另外一个字节的映射。S盒的详细构造方法可以参考文献[1]。这里直接给出构造好的结果，下图(a)为S盒，图(b)为S^{-1}（S盒的逆）。

		y														
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(a) S-box

x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

(b) Inverse S-box

S和S^{-1}分别为16x16的矩阵。输入的高4-bit对应的值作为行标，低4-bit对应的值作为列标；假设输入字节的值为$a=a_7a_6a_5a_4a_3a_2a_1a_0$，则输出值为$S[a_7a_6a_5a_4][a_3a_2a_1a_0]$，$S^{-1}$的变换也同理。

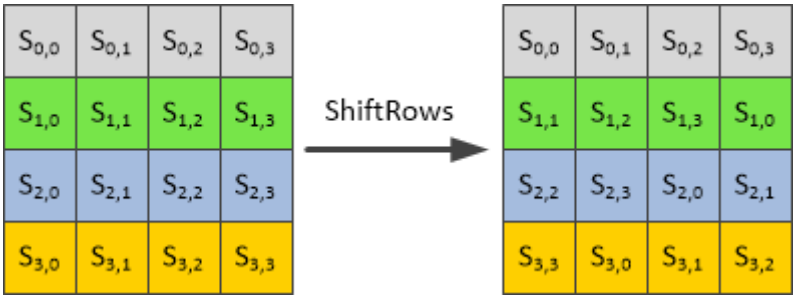
例如：字节00000000_B替换后的值为（S[0][0]=）63_H，再通过S^{-1}即可得到替换前的值，（S^{-1} [6][3]=）00_H。

1.2 行移位

行移位的功能是实现一个4x4矩阵内部字节之间的置换。

1.2.1 正向行移位

正向行移位的原理图如下：



实际移位的操作即是：第一行保存不变，第二行循环左移1个字节，第三行循环左移2个字节，第四行循环左移3个字节。假设矩阵的名字为state，用公式表示如下：state'[i][j] = state[i][(j+i)%4];其中i、j属于[0,3]

1.2.2 逆向行移位

逆向行移位即是相反的操作，用公式表示如下：state'[i][j] = state[i][(4+j-i)%4];其中i、j属于[0,3]

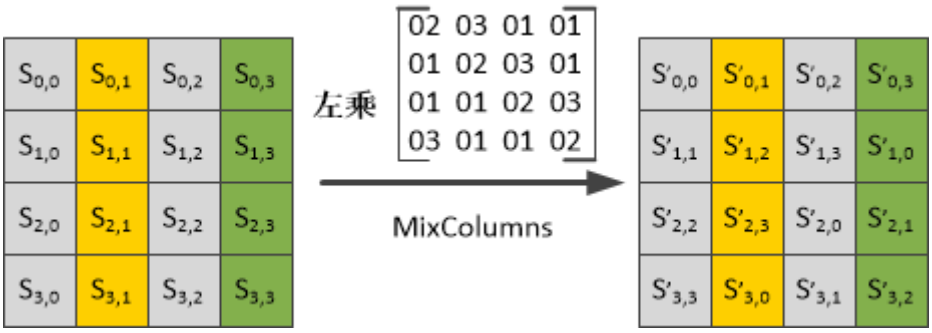
到这里差不多完成一半了.....

1.3 列混淆

列混淆：利用GF(2^8)域上算术特性的一个代替。

1.3.1 正向列混淆

正向列混淆的原理图如下：



根据矩阵的乘法可知，在列混淆的过程中，每个字节对应的值只与该列的4个值有关系。此处的乘法和加法都是定义在GF(2^8)上的，需要注意如下几点：

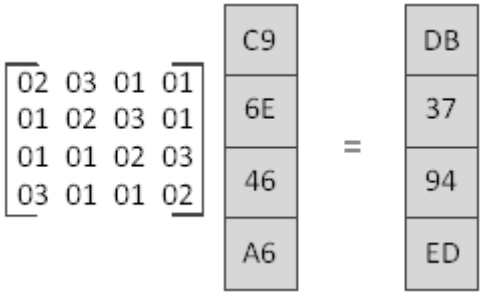
1) 将某个字节所对应的值乘以2，其结果就是将该值的二进制位左移一位，如果原始值的最高位为1，则还需要将移位后的结果异或00011011；[1]

英文原文描述如下：In particular, multiplication of a value by x (i.e., by {02}) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with (0001 1011) if the leftmost bit of the original value (prior to the shift) is 1.

2) 乘法对加法满足分配率，例如：$07 \cdot S_{0,0}=(01 \oplus 02 \oplus 04) \cdot S_{0,0}= S_{0,0} \oplus (02 \cdot S_{0,0}) (04 \cdot S_{0,0})$

3) 此处的矩阵乘法与一般意义上矩阵的乘法有所不同，各个值在相加时使用的是模2加法（异或运算）。

下面举一个例子，假设某一列的值如下图，运算过程如下：



$$S'_{0,0} = (02 \bullet C9) \oplus (03 \bullet 6E) \oplus (01 \bullet 46) \oplus (01 \bullet A6)$$

其中：

$$02 \bullet C9 = 02 \bullet 11001001_B = 10010010_B \oplus 00011011_B = 10001001_B$$

$$03 \bullet 6E = (01 \oplus 02) \bullet 6E = 01101110_B \oplus 11011100_B = 10110010_B$$

$$01 \bullet 46 = 01000110_B$$

$$01 \bullet A6 = 10100110_B$$

则：

$$S'_{0,0} = 10001001_B \oplus 10110010_B \oplus 01000110_B \oplus 10100110_B$$

$$= 11011011_B = DB$$

在计算02与C9的乘积时，由于C9对应最左边的比特为1，因此需要将C9左移一位后的值与(0001 1011)求异或。同理可以求出另外几个值。

1.3.2 逆向列混淆

逆向列混淆的原理图如下：

由于：

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \\ 00 & 00 & 00 & 01 \end{bmatrix}$$

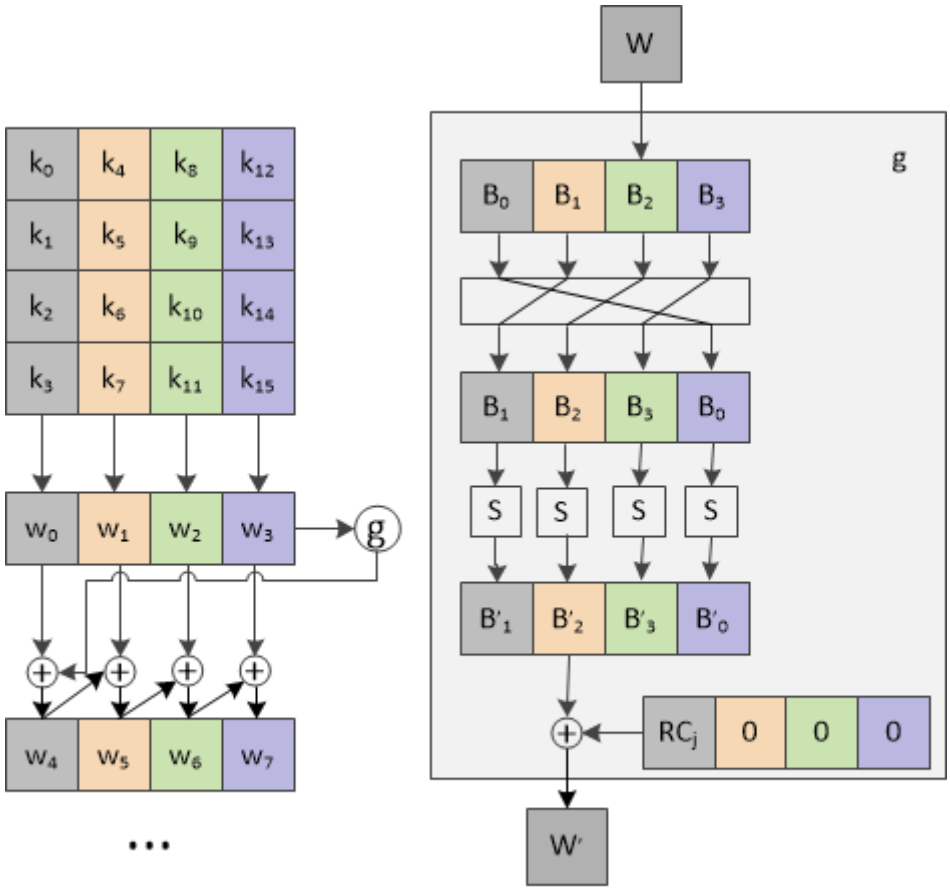
说明两个矩阵互逆，经过一次逆向列混淆后即可恢复原文。

1.4 轮密码加

这个操作相对简单，其依据的原理是“任何数和自身的异或结果为0”。加密过程中，每轮的输入与轮密钥异或一次；因此，解密时再异或上该轮的密钥即可恢复输入。

1.5 密钥扩展

密钥扩展的原理图如下：



(a) 总体算法　　(b) 函数g

密钥扩展过程说明：

1）将初始密钥以列为主，转化为4个32 bits的字，分别记为w[0...3]；

2）按照如下方式，依次求解w[j]，其中j是整数并且属于[4,43]；

3）若j%4=0,则w[j]=w[j-4]⊕g(w[j-1]),否则w[j]=w[j-4]⊕w[j-1]；

函数g的流程说明：

4）将w循环左移一个字节；

5）分别对每个字节按S盒进行映射；

6）与32 bits的常量（RC[j/4],0,0,0）进行异或，RC是一个一维数组，其值如下。（RC的值只需要有10个，而此处用了11个，实际上RC[0]在运算中没有用到，增加RC[0]是为了便于程序中用数组表示。由于j的最小取值是4，j/4的最小取值则是1，因此不会产生错误。）

RC = {00, 01, 02, 04, 08, 10, 20, 40, 80, 1B, 36}

好吧，整个加密过程就是这样，终于介绍完了……

解密过程直接进行逆操作即可，但是为了减少硬件面积，可以通过对解密的操作进行适当调换，再使用与加密相同的硬件电路。

1.6 小结

密码算法要求是可逆的，这样解密才能正确的恢复明文。拿AES来说，在密钥固定的情况下，明文和密文在整个输入空间是一一对应的。因此算法的各个部件也都是可逆的，再将各个部件的操作顺序设计成可逆的，密文就能正确的解密了。

2 源码

自己写了一份AES-128的实现代码，放在Github上；另外一份AES代码实现了3种密钥长度的算法，有兴趣可以看看。

3 参考

[1] William Stallings著；王张宜等译. 密码编码学与网络安全——原理与实践（第五版）[M]. 北京：电子工业出版社，2012.1.

[2] Daemen J, Rijmen V. AES proposal: Rijndael[J]. 1998.

[3] Advanced Encryption Standard Process.

密码算法详解——AES的更多相关文章

1. 密码算法详解——DES
   0 DES简介 在20世纪60年代后期,IBM公司成立了一个由Horst Feistel负责的计算机密码学研究项目.1971年设计出密码算法LUCIFER后,该项目宣告结束.LUCIFER被卖给了伦敦 ...
2. 密码算法详解——Simon
   0 Simon简介 详细文档请直接阅读参考文献[1]. Simon是由NSA设计的轻量级分组密码算法(LIGHTWEIGHT BLOCK CIPHER).主要应用于硬件或软件条件受限(例如:芯片面积要 ...
3. 信息安全－1：python之playfair密码算法详解[原创]
   转发注明出处: http://www.cnblogs.com/0zcl/p/6105825.html 一.基本概念 古典密码是基于字符替换的密码.加密技术有:Caesar(恺撒)密码.Vigenere ...
4. BM算法　　Boyer-Moore高质量实现代码详解与算法详解
   Boyer-Moore高质量实现代码详解与算法详解 鉴于我见到对算法本身分析非常透彻的文章以及实现的非常精巧的文章,所以就转载了,本文的贡献在于将两者结合起来,方便大家了解代码实现! 算法详解转自:h ...
5. kmp算法详解
   转自:http://blog.csdn.net/ddupd/article/details/19899263 KMP算法详解 KMP算法简介: KMP算法是一种高效的字符串匹配算法,关于字符串匹配最简 ...
6. 机器学习经典算法详解及Python实现--基于SMO的SVM分类器
   原文:http://blog.csdn.net/suipingsp/article/details/41645779 支持向量机基本上是最好的有监督学习算法,因其英文名为support vector ...
7. [转] KMP算法详解
   转载自:http://www.matrix67.com/blog/archives/115 KMP算法详解 如果机房马上要关门了,或者你急着要和MM约会,请直接跳到第六个自然段.　我们这里说的K ...
8. 【转】AC算法详解

原文转自:http://blog.csdn.net/joylnwang/article/details/6793192 AC算法是Alfred V.Aho(←编译原理→)(龙书)的作者),和 ...

9. KMP算法详解(转自中学生OI写的。。ORZ！)

KMP算法详解 如果机房马上要关门了,或者你急着要和MM约会,请直接跳到第六个自然段. 我们这里说的KMP不是拿来放电影的(虽然我很喜欢这个软件),而是一种算法.KMP算法是拿来处理字符串匹配的.换句 ...

随机推荐

1. hibernate优化笔记(随时更新)

一:优化配置 1.关联映射的配置:对照之前的博客,如:inverse属性的设置(减少对同一对象的多条update语句):在one端设置为true,只会执行一次update语句 2.级联cascade属 ...

2. SpringMvc面试题

f-sm-1. 讲下SpringMvc和Struts1,Struts2的比较的优势 性能上 Struts1→SpringMvc→Struts2 开发速度上SpringMvc和Struts2差 ...

3. （转）WCF开发框架形成之旅---WCF的几种寄宿方式

WCF寄宿方式是一种非常灵活的操作,可以在IIS服务.Windows服务.Winform程序.控制台程序中进行寄宿,从而实现WCF服务的运行,为调用者方便.高效提供服务调用.本文分别对这几种方式进行详 ...

4. Logstash 安装与配置

一.Logstash 描述 简单而又强大的数据抽取与处理工具,相比于flums一整本书的描述强大而又好用. 还记得我13年用python写了一个数据抽取.校验工具,设计思路也同样是拆解处理过程模板,然 ...

5. php file_get_contents失败[function.file-get-contents]: failed to open stream: HTTP request failed!解决

在使用file_get_contents方法来获取远程文件时会出现 [function.file-get-contents]: failed to open stream: HTTP request ...

6. 第1个linux驱动___打印"hello world"

为了方便后续的深入,我们在驱动程序中用printk( )函数来打印"hello world",printk( )是内核中自带的函数,专门用于在打印内核信息.在安装驱动模块到内核中的 ...

7. 一次关于使用status作为变量引发的bug及思考

这个bug出现在一年前,当时自己大学还没毕业,刚刚进入一家公司实习.那个时候还没有用seajs或者requirejs那样的模块化管理的库,也没有用一个自执行的函数将要执行的代码包裹起来,于是bug就在 ...

8. WEB安全--CSRF防御

   CSRF漏洞防御主要可以从三个层面进行,即服务端的防御.用户端的防御和安全设备的防御. 服务端的防御 目前服务器端防御CSRF攻击主要有5种策略(我知道的就这么多):验证HTTP Referer字段, ...

9. Oracle查看表结构的几种方法

   1,DESCRIBE 命令 使用方法如下: SQL→ describe nchar_tst(nchar_tst为表名) 显示的结果如下: 名称 ...

10. Linux面试基础题-2

    继续我们这面试系列,在这篇文章里我们给出了10个问题.这些问题或者是在以后的文章中出现的问题不一定在面试中会被问到.然而通过这些文章我们呈现出的是一个交互的学习平台,这必将会对你有很大的帮助. 自本系 ...