

个人资料



Arrow

访问：2116978次

积分：22338

等级：BLOG > 7

排名：第205名

原创：268篇

转载：183篇

译文：5篇

评论：333条

文章搜索

文章分类

Linux驱动 (29)

Android OpenGL (0)

Android应用 (44)

Linux Kernel (48)

Android Media (10)

Android Framework (27)

Stagefright (8)

DisplaySystem (8)

HAL (2)

CPU&GPU (18)

HW (21)

Android系统 (50)

Android调试 (8)

基础知识 (56)

OpenGL ES (12)

项目管理 (4)

测试 (4)

Google Play Store (2)

工作规范 (1)

Audio (3)

Linux内存管理--基本概念

2013-03-01 09:2911484人阅读评论(9)收藏举报

分类：Linux Kernel (47)内存管理 (3)

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录(?)

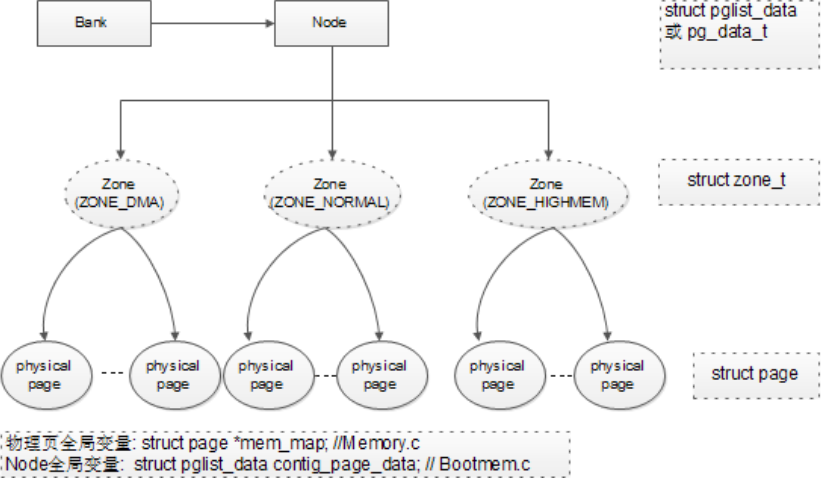
1. Linux物理内存三级架构

Linux数据结构

struct pglist\_data或pg\_data\_t

struct zone\_t

struct page



对于内存管理，Linux采用了与具体体系架构不相关的设计模型，实现了良好的可伸缩性。它主要由内存节点node、内存区域zone和物理页框page三级架构组成。

内存节点node

内存节点node是计算机系统中对物理内存的一种描述方法，一个总线主设备访问位于同一个节点中的任意内存单元所花的代价相同，而访问任意两个不同节点中的内存单元所花的代价不同。在一致存储结构(Uniform Memory Architecture，简称UMA)计算机系统中只有一个节点，而在非一致性存储结构(NUMA)计算机系统中有多节点。Linux内核中使用数据结构pg\_data\_t来表示内存节点node。如常用的ARM架构为UMA架构。

内存区域zone

内存区域位于同一个内存节点之内，由于各种原因它们的用途和使用方法并不一样。如基于IA32体系结构的个人计算机系统中，由于历史原因使得ISA设备只能使用最低16MB来进行DMA传输。又如，由于Linux内核采用

无线通信 (1)

Android基础知识 (21)

认证 (7)

商品管理 (1)

Android OTA (6)

Android WiFi (12)

Android待机唤醒 (8)

ALSA (13)

V4L2&USB (14)

内存管理 (4)

Linux API (2)

Linux安全 (9)

云计算 (1)

异构计算 (1)

cocos2d-x (17)

Unity3D (25)

H.265/HEVC (4)

3DS Max (3)

Photoshop (1)

BigData (19)

读书 (1)

思想 (0)

AI (25)

文章存档

2016年06月 (4)

2016年05月 (10)

2016年04月 (14)

2016年03月 (14)

2016年01月 (5)

展开

阅读排行

WiFi基本知识 (61465)

Linux pipe函数 (44823)

Android 4.0 事件输入(Ev (38894)

SELinux深入理解 (30188)

Linux inotify功能及实现原 (29063)

Android WiFi--系统架构 (28822)

Android4.0.3 显示系统深 (28650)

Android App 隐藏标题栏 (27881)

Android4.x 如何处理Pow (26415)

USB枚举过程 (25167)

评论排行

Android4.0.3 显示系统深 (25)

Android 4.0 事件输入(Ev (21)

WiFi基本知识 (19)

USB协通讯议--深入理解 (14)

Android4.x 如何处理Pow (12)

- 物理页框page

2. Linux虚拟内存三级页表

Linux虚拟内存三级管理由以下三级组成：

- PGD: Page Global Directory (页目录)
- PMD: Page Middle Directory (页目录)
- PTE: Page Table Entry (页表项)

每一级有以下三个关键描述宏：

- SHIFT
- SIZE
- MASK

如页的对应描述为：

```
[cpp]
01.  /* PAGE_SHIFT determines the page size  asm/page.h */
02.  #define PAGE_SHIFT      12
03.  #define PAGE_SIZE      (_AC(1,UL) << PAGE_SHIFT)
04.  #define PAGE_MASK      (~(PAGE_SIZE-1))
```

数据结构定义如下：

```
[cpp]
01.  /* asm/page.h */
02.  typedef unsigned long pteval_t;
03.
04.  typedef pteval_t pte_t;
05.  typedef unsigned long pmd_t;
06.  typedef unsigned long pgd_t[2];
07.  typedef unsigned long pgprot_t;
08.
09.  #define pte_val(x)      (x)
10.  #define pmd_val(x)      (x)
11.  #define pgd_val(x)      ((x)[0])
12.  #define pgprot_val(x)   (x)
13.
14.  #define __pte(x)        (x)
15.  #define __pmd(x)        (x)
16.  #define __pgprot(x)     (x)
```

2.1 Page Directory (PGD and PMD)

每个进程有它自己的PGD( Page Global Directory)，它是一个物理页，并包含一个pgd\_t数组。其定义见<asm/page.h>。进程的pgd\_t数据见 task\_struct -> mm\_struct -> pgd\_t \* pgd;

ARM架构的PGD和PMD的定义如下<arch/arm/include/asm/pgtable.h>：

Linux Wireless架构总结	(11)
Android ActivityThread(三)	(10)
Unity5.0与Android交互	(10)
Linux内存管理--基本概念	(9)
USB枚举过程	(9)

推荐文章

\*浅谈android中异步加载之"取消异步加载"二

\*笑谈Android图表-----MPAndroidChart

\*Nginx正向代理、负载均衡等功能实现配置

\*浅析ZeroMQ工作原理及其特点

\*Android官方开发文档Training系列课程中文版：多样屏幕的支持不同的屏幕尺寸

\*Spring Boot 实践折腾记（三）：三板斧，Spring Boot下使用Mybatis

最新评论

Android热插拔事件处理流程--Vodengziliang001: 挂载流程的第6点应该有问题吧，DirectVolume::handleDiskAdded()的消息是...

Android ActivityThread(主线程或DangerYang: @yuwang\_00:我觉得也是写反了

SELinux策略语言--客体类别和许DGerome: 各许可的含义有点抽象，要是实例就好了，比如对于文件而言的五种SELinux特定许可：relabel...

Linux如何查看与/dev/input目录下的网络人VS灰鸽子: 直接用代码动态获取  
http://blog.csdn.net/qq\_21792169/a

WiFi基本知识  
tinylaker: 博主，文章我转载收藏了

WiFi信号强度--SIGNAL\_POLL  
PineappleMushroom: 同问 这是什么软件画的 谢谢

MySQL基本知识  
Arrow: 自己建一个环境，做一个小case，遇到问题就Baidu，基本上所有相关知识都熟悉了

MySQL基本知识  
Cherry???: 你好，我是数据库MySQL初学者，请问怎样可以快速上手MySQL

深入了解MediaServer-1  
flaoter: code能不能好好排一下版？

android系统关机流程  
lml1010402004: that is very nice.

[cpp]C?01. <p>#define PTRS\_PER\_PTE 512 // PTE中可包含的指针<u32>数 (21-12=9bit)02. #define PTRS\_PER\_PMD 103. #define PTRS\_PER\_PGD 2048 // PGD中可包含的指针<u32>数 (32-21=11bit)</p><p>#define PTE\_HWTABLE\_PTRS (PTRS\_PER\_PTE)04. #define PTE\_HWTABLE\_OFF (PTE\_HWTABLE\_PTRS \* sizeof(pte\_t))05. #define PTE\_HWTABLE\_SIZE (PTRS\_PER\_PTE \* sizeof(u32))</p><p>/\*06. \* PMD\_SHIFT determines the size of the area a second-level page table can map07. \* PGDIR\_SHIFT determines what a third-level page table entry can map08. \*/09. #define PMD\_SHIFT 2110. #define PGDIR\_SHIFT 21</p>11. <span style="font-size:18px;">虚拟地址SHIFT宏图 :</span>

Linear Address

BITS\_PER\_LONG

Global (PGD)

Middle (PMD)

Table (PTE)

Offset

PAGE\_SHIFT

PMD\_SHIFT

PGDIR\_SHIFT

Linear Address Bit Size Macros

虚拟地址MASK和SIZE宏图：



## 2.2 Page Table Entry

PTEs, PMDs和PGDs分别由pte\_t, pmd\_t和pgd\_t来描述。为了存储保护位，pgprot\_t被定义，它拥有相关的flags并经常被存储在page table entry低位(lower bits)，其具体的存储方式依赖于CPU架构。

每个pte\_t指向一个物理页的地址，并且所有的地址都是页对齐的。因此在32位地址中有PAGE\_SHIFT(12)位是空闲的，它可以为PTE的状态位。

PTE的保护和状态位如下图所示：

Bit	Function
_PAGE_PRESENT	Page is resident in memory and not swapped out
_PAGE_PROTNONE	Page is resident but not accessable
_PAGE_RW	Set if the page may be written to
_PAGE_USER	Set if the page is accessible from user space
_PAGE_DIRTY	Set if the page is written to
_PAGE_ACCESSED	Set if the page is accessed

Page Table Entry Protection and Status Bits

## 2.3 如何通过3级页表访问物理内存

为了通过PGD、PMD和PTE访问物理内存，其相关宏在asm/pgtable.h中定义。

#### • pgd\_offset

根据当前虚拟地址和当前进程的mm\_struct获取pgd项的宏定义如下：

```
[cpp] C ?
01. /* to find an entry in a page-table-directory */
02. #define pgd_index(addr) ((addr) >> PGDIR_SHIFT) //获得在pgd表中的索引
03.
04. #define pgd_offset(mm, addr) ((mm)->pgd + pgd_index(addr)) //获得pmd表的起始地址
05.
06. /* to find an entry in a kernel page-table-directory */
07. #define pgd_offset_k(addr) pgd_offset(&init_mm, addr)
```

#### • pmd\_offset

根据通过pgd\_offset获取的pgd项和虚拟地址，获取相关的pmd项(即pte表的起始地址)

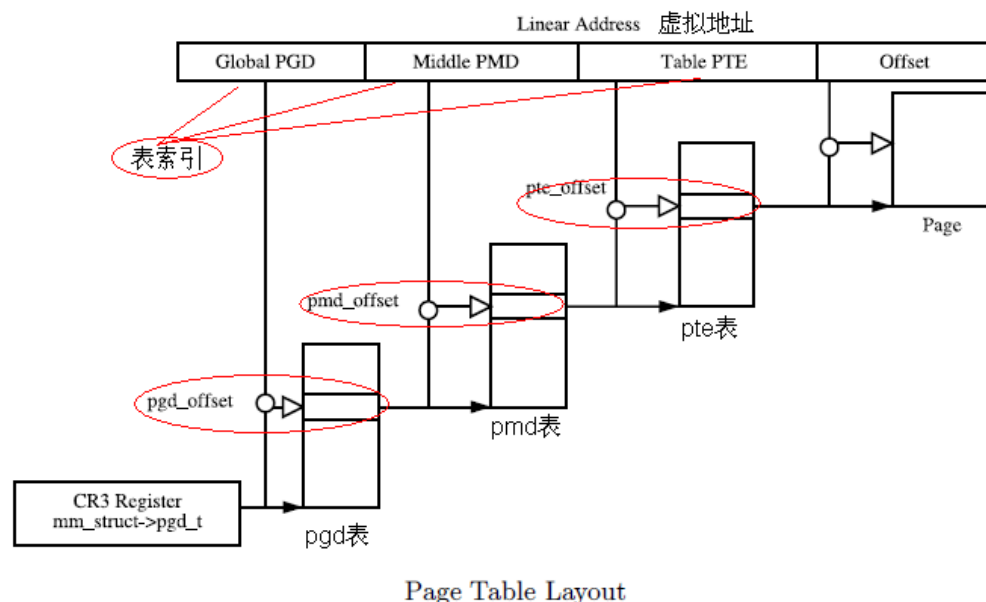
```
[cpp] C ?
01. /* Find an entry in the second-level page table.. */
02. #define pmd_offset(dir, addr) ((pmd_t *) (dir)) //即为pgd项的值
```

#### • pte\_offset

根据通过pmd\_offset获取的pmd项和虚拟地址，获取相关的pte项(即物理页址)

```
[cpp] C ?
01. #ifndef CONFIG_HIGHPTE
02. #define __pte_map(pmd) pmd_page_vaddr(*(pmd))
03. #define __pte_unmap(pte) do { } while (0)
04. #else
05. #define __pte_map(pmd) (pte_t *)kmap_atomic(pmd_page(*(pmd)))
06. #define __pte_unmap(pte) kunmap_atomic(pte)
07. #endif
08.
09. #define pte_index(addr) (((addr) >> PAGE_SHIFT) & (PTRS_PER_PTE - 1))
10.
11. #define pte_offset_kernel(pmd, addr) (pmd_page_vaddr(*(pmd)) + pte_index(addr))
12.
13. #define pte_offset_map(pmd, addr) (__pte_map(pmd) + pte_index(addr))
14. #define pte_unmap(pte) __pte_unmap(pte)
15.
16. #define pte_pfn(pte) (pte_val(pte) >> PAGE_SHIFT)
17. #define pfn_pte(pfn, prot) __pte(__pfn_to_phys(pfn) | pgprot_val(prot))
18.
19. #define pte_page(pte) pfn_to_page(pte_pfn(pte))
20. #define mk_pte(page, prot) pfn_pte(page_to_pfn(page), prot)
21.
22. #define set_pte_ext(pte, pte_val, ext) cpu_set_pte_ext(pte, pte_val, ext)
23. #define pte_clear(mm, addr, ptep) set_pte_ext(ptep, __pte(0), 0)
```

其示意图如下图所示：



## 2.4 根据虚拟地址获取物理页的示例代码

根据虚拟地址获取物理页的示例代码详见<mm/memory.c中的函数follow\_page>。

```
[cpp]
01. /**
02.  * follow_page - look up a page descriptor from a user-virtual address
03.  * @vma: vm_area_struct mapping @address
04.  * @address: virtual address to look up
05.  * @flags: flags modifying lookup behaviour
06.  *
07.  * @flags can have FOLL_ flags set, defined in <linux/mm.h>
08.  *
09.  * Returns the mapped (struct page *), %NULL if no mapping exists, or
10.  * an error pointer if there is a mapping to something not represented
11.  * by a page descriptor (see also vm_normal_page()).
12.  */
13. struct page *follow_page(struct vm_area_struct *vma, unsigned long address,
14.                          unsigned int flags)
15. {
16.     pgd_t *pgd;
17.     pud_t *pud;
18.     pmd_t *pmd;
19.     pte_t *ptep, pte;
20.     spinlock_t *ptl;
21.     struct page *page;
22.     struct mm_struct *mm = vma->vm_mm;
23.
24.     page = follow_huge_addr(mm, address, flags & FOLL_WRITE);
25.     if (!IS_ERR(page)) {
26.         BUG_ON(flags & FOLL_GET);
27.         goto out;
28.     }
29.
30.     page = NULL;
31.     pgd = pgd_offset(mm, address);
32.     if (pgd_none(*pgd) || unlikely(pgd_bad(*pgd)))
33.         goto no_page_table;
34.
35.     pud = pud_offset(pgd, address);
36.     if (pud_none(*pud))
37.         goto no_page_table;
38.     if (pud_huge(*pud) && vma->vm_flags & VM_HUGETLB) {
```

```

39.         BUG_ON(flags & FOLL_GET);
40.         page = follow_huge_pud(mm, address, pud, flags & FOLL_WRITE);
41.         goto out;
42.     }
43.     if (unlikely(pud_bad(*pud)))
44.         goto no_page_table;
45.
46.     pmd = pmd_offset(pud, address);
47.     if (pmd_none(*pmd))
48.         goto no_page_table;
49.     if (pmd_huge(*pmd) && vma->vm_flags & VM_HUGETLB) {
50.         BUG_ON(flags & FOLL_GET);
51.         page = follow_huge_pmd(mm, address, pmd, flags & FOLL_WRITE);
52.         goto out;
53.     }
54.     if (pmd_trans_huge(*pmd)) {
55.         if (flags & FOLL_SPLIT) {
56.             split_huge_page_pmd(mm, pmd);
57.             goto split_fallthrough;
58.         }
59.         spin_lock(&mm->page_table_lock);
60.         if (likely(pmd_trans_huge(*pmd))) {
61.             if (unlikely(pmd_trans_splitting(*pmd))) {
62.                 spin_unlock(&mm->page_table_lock);
63.                 wait_split_huge_page(vma->anon_vma, pmd);
64.             } else {
65.                 page = follow_trans_huge_pmd(mm, address,
66.                                             pmd, flags);
67.                 spin_unlock(&mm->page_table_lock);
68.                 goto out;
69.             }
70.         } else
71.             spin_unlock(&mm->page_table_lock);
72.         /* fall through */
73.     }
74. split_fallthrough:
75.     if (unlikely(pmd_bad(*pmd)))
76.         goto no_page_table;
77.
78.     ptep = pte_offset_map_lock(mm, pmd, address, &ptl);
79.
80.     pte = *ptep;
81.     if (!pte_present(pte))
82.         goto no_page;
83.     if ((flags & FOLL_WRITE) && !pte_write(pte))
84.         goto unlock;
85.
86.     page = vm_normal_page(vma, address, pte);
87.     if (unlikely(!page)) {
88.         if ((flags & FOLL_DUMP) ||
89.             !is_zero_pfn(pte_pfn(pte)))
90.             goto bad_page;
91.         page = pte_page(pte);
92.     }
93.
94.     if (flags & FOLL_GET)
95.         get_page(page);
96.     if (flags & FOLL_TOUCH) {
97.         if ((flags & FOLL_WRITE) &&
98.             !pte_dirty(pte) && !PageDirty(page))
99.             set_page_dirty(page);
100.    /*
101.     * pte_mkyoung() would be more correct here, but atomic care
102.     * is needed to avoid losing the dirty bit: it is easier to use
103.     * mark_page_accessed().
104.     */
105.    mark_page_accessed(page);
106.    }
107.    if ((flags & FOLL_MLOCK) && (vma->vm_flags & VM_LOCKED)) {
108.        /*
109.         * The preliminary mapping check is mainly to avoid the
110.         * pointless overhead of lock_page on the ZERO_PAGE

```

```
111.         * which might bounce very badly if there is contention.
112.         *
113.         * If the page is already locked, we don't need to
114.         * handle it now - vmscan will handle it later if and
115.         * when it attempts to reclaim the page.
116.         */
117.         if (page->mapping && trylock_page(page)) {
118.             lru_add_drain(); /* push cached pages to LRU */
119.             /*
120.              * Because we lock page here and migration is
121.              * blocked by the pte's page reference, we need
122.              * only check for file-cache page truncation.
123.              */
124.             if (page->mapping)
125.                 mlock_vma_page(page);
126.             unlock_page(page);
127.         }
128.     }
129.     unlock:
130.         pte_unmap_unlock(pte, ptl);
131.     out:
132.         return page;
133.
134.     bad_page:
135.         pte_unmap_unlock(pte, ptl);
136.         return ERR_PTR(-EFAULT);
137.
138.     no_page:
139.         pte_unmap_unlock(pte, ptl);
140.         if (!pte_none(pte))
141.             return page;
142.
143.     no_page_table:
144.         /*
145.          * When core dumping an enormous anonymous area that nobody
146.          * has touched so far, we don't want to allocate unnecessary pages or
147.          * page tables. Return error instead of NULL to skip handle_mm_fault,
148.          * then get_dump_page() will return NULL to leave a hole in the dump.
149.          * But we can only make this optimization where a hole would surely
150.          * be zero-filled if handle_mm_fault() actually did handle it.
151.          */
152.         if ((flags & FOLL_DUMP) &&
153.             (!vma->vm_ops || !vma->vm_ops->fault))
154.             return ERR_PTR(-EFAULT);
155.         return page;
156.     }
```

顶

1

踩

0

上一篇

machine\_desc结构体

下一篇

Linux内存管理--物理内存分配

我的同类文章

Linux Kernel ( 47 )	内存管理 ( 3 )
---------------------	------------



• 用户态应用程序直接与USB...	2013-10-29	阅读 1660	• Linux输入子系统：事件的编...	2013-10-24	阅读 1064
• 读取并显示/dev/input/event...	2013-10-08	阅读 2619	• Linux /dev/uinput	2013-09-27	阅读 7176
• Linux如何查看与/dev/input目..	2013-08-30	阅读 14328	• SELinux - Multi-Level Secur...	2013-08-12	阅读 2720
• SELinux - MCS	2013-08-12	阅读 1267	• Linux安全--访问控制机制(A...	2013-08-12	阅读 4189
• SELinux深入理解	2013-08-09	阅读 30188	• SELinux简介	2013-08-08	阅读 3802
• Linux Socket编程	2013-08-07	阅读 1163			
更多文章					

猜你在找

高并发集群架构超细精讲	Linux内存管理--基本概念
数据结构（C版）	linux内存管理伙伴算法—基本概念介绍
嵌入式工程师养成计划之——嵌入式软件工程师完全学	Linux内存管理之一基本概念篇
数据结构基础系列(6)：树和二叉树	Linux内存管理的基本概念
微信公众平台开发入门	Linux内存管理及其基本概念

查看评论

6楼 [大圣喜欢吃香蕉](#) 2015-11-17 23:36发表



清楚了。

5楼 [lieye\\_leaves](#) 2014-07-10 22:11发表



```
>#define PTRS_PER_PTE 512 // PTE中可包含的指针<u32>数 (21-12=9bit)
02.#define PTRS_PER_PMD 1
03.#define PTRS_PER_PGD 2048 // PGD中可包含的指针<u32>数 (32-21=11bit)</pre>
```

你好，向博主提个问题，文中提到>#define PTRS\_PER\_PTE =512  
PTRS\_PER\_PGD =2048  
那么虚拟地址的组成可以这样理解  
0000 0000 000 | 0 0000 0000 | 0000 0000 0000  
11位 9位 12位  
PGD PTE 页内偏移  
但是一般的MMU，他的位数是 12位+8位+12位，但是linux arm如上所示是11位+9位+12位，这该如何理解，MMU是怎么转换的。

4楼 [罗万千](#) 2014-06-18 16:32发表



好文，书上讲解的有点不理解，看到此文懂了很多，谢谢楼主

3楼 [yjpcn](#) 2014-01-27 17:05发表



看了这么多天，终于有些理解了，follow\_page只是根据虚拟地址找到描述对应物理地址所在page的结构体，返回的是指向这个结构体的指针，也是个虚拟地址。  
原来理解成找到对应的物理地址了。

2楼 [yjpcn](#) 2013-12-16 10:22发表



虚拟地址不就是米吗？  
在arm的手册上对MMU的功能是这么说的：It supports address translation from an input address to an output address, based on address mapping and memory attribute information held in translation tables.  
那么input address不就是虚拟地址吗？  
软件中要把映射表配置好。

1楼 [yjpcn](#) 2013-12-13 15:53发表



MMU不上负责把虚拟地址转成物理地址的吗？  
为什么软件还要做这个东西呢？？

Re: [Arrow](#) 2013-12-13 16:43发表

回复yjpcn：MMU就是一个高级厨师，没米也无法做饭啊



Re: [duanlove](#) 2013-12-15 21:30发表



回复MyArrow：内存分配那一块与 MMU/页表 虚拟地址到物理地址的映射的关系你熟悉么？

Re: [my0929my](#) 2015-10-22 15:13发表



回复duanlove：MMU 配置，其实就是将物理地址偏移和虚拟地址的偏移，以及对应的符号表的映射关系对应起来！

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

\* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

- 全部主题
- Hadoop    AWS    移动游戏    Java    Android    iOS    Swift    智能硬件    Docker    OpenStack
- VPN    Spark    ERP    IE10    Eclipse    CRM    JavaScript    数据库    Ubuntu    NFC    WAP    jQuery
- BI    HTML5    Spring    Apache    .NET    API    HTML    SDK    IIS    Fedora    XML    LBS    Unity
- Splashtop    UML    components    Windows Mobile    Rails    QEMU    KDE    Cassandra    CloudStack    FTC
- coremail    OPhone    CouchBase    云计算    iOS6    Rackspace    Web App    SpringSide    Maemo
- Compuware    大数据    aptech    Perl    Tornado    Ruby    Hibernate    ThinkPHP    HBase    Pure    Solr
- Angular    Cloud Foundry    Redis    Scala    Django    Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服    杂志客服    微博客服    [webmaster@csdn.net](mailto:webmaster@csdn.net)    400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2014, CSDN.NET, All Rights Reserved