qemu如何实现面向对象模型QOM（代码讲解）

原创 YuanruiZJU 最后发布于2016-12-12 12:41:11 阅读数 1808 ☆ 收藏

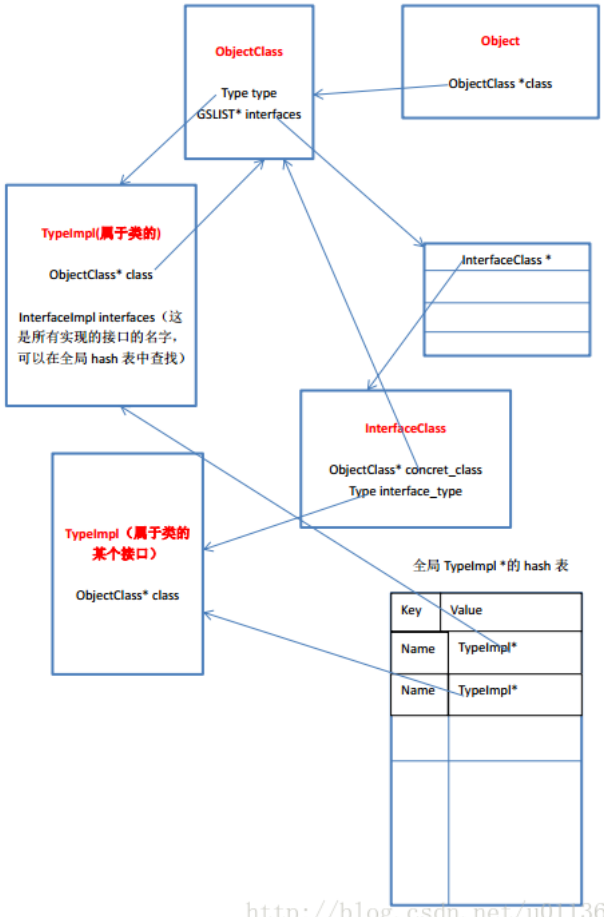
（说明：本文档只是对QOM中关键实现片段进行叙述，更加详细的代码，请查看本文涉及的代码文件）
有两个问题需要解答：

1. QOM中如何将所有的类储存起来的，并且完整地呈现给使用者的。
2. 面向对象编程，有三个重要的特性——封装、继承和多态。封装可以隐藏实现细节，使得代码模块化；继承可以扩展已存在的代码模块（类）是为了代码重用。而多态则是为了实现另一个目的——接口重用。多态的作用，就是为了类在继承和派生的时候，保证使用“基类”中任一类性时的正确调用。本文档将对QOM中实现的封装、继承和多态三个特性分别展开叙述。

    1. 封装
       封装就是将数据或函数等集合在一个个的单元中（我们称之为类）。被封装的对象通常被称为抽象数据类型。
    2. 继承
       继承是指这样一种能力：它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展。
       通过继承创建的新类称为“子类”或“派生类”。被继承的类称为“基类”、“父类”或“超类”。
       继承的过程，就是从一般到特殊的过程。
    3. 多态
       多态是指同一操作作用于不同的对象，可以有不同的解释，产生不同的执行结果。在运行时，可以通过指向基类的指针，来调用实现派

整体把握几个数据结构

下面这张图可以简要说明ObjectClass、Object、TypeImpl、InterfaceClass之间的关系。通过这些数据结构中的指针，我们可以很方便地找到其构。



QOM如何将所有的类储存起来的

用户如何通过QOM提供的数据结构表示一个类?

在我们之前对QOM的介绍篇中已经说明了，在qemu中如何创建一个新的类。用户在创建类时要实现两个数据结构：描述类的数据结构的实例）、描述类产生的实例的数据结构（第一个属性必须是父类对象的数据结构的实例），并且需要创建一个数据结构TypeInfo的实例，的类型。

通过创建TypeInfo，可以告诉qemu哪些信息呢?
我们再次查看一下TypeInfo数据结构，这个数据结构中包含了这个类型的父类的名字，insatnce的size信息、class的size信息，instance的初始化class的初始化和垃圾回收函数。有了这些信息我们就可以：
* 为instance、class初始化第一个属性（instance的第一个属性必须是父类对象的数据结构的实例，class的一个属性必须是父类的结构的实例
* 为instance、class分配合适的内存空间，这样我们就可以将一个Object的指针动态cast为我们需要的对象类型。
* 类的数据结构一般包含大量的函数指针，用于对对象进行操作。class的init函数可以将这些函数指针初始化。然后所有含有这个数据结构指针的用这些函数。
* 对象的数据结构一般包含了大量变量，是对象的一些属性。instance的init函数可以把这些属性初始化，相当于C++中的构造函数
* 一个类可以实现多个接口，这些接口也是一个类，并且是抽象类，含有虚拟函数指针。

可以说，有了这个数据结构，就有了类的最基本的信息。（代码在include/qom/object.h中）

```
struct TypeInfo
{
    const char *name; //这个类型的名字
    const char *parent;  //这个类型的父类的名字

    size_t instance_size; //对象对应数据结构的size

    // instance如何初始化和最后的垃圾回收
    void (*instance_init)(Object *obj);
    void (*instance_post_init)(Object *obj);
    void (*instance_finalize)(Object *obj);

    bool abstract; //这个类是否是抽象的，也就是是否有虚拟函数
    size_t class_size; //类对应数据结构的size

    // 类如何初始化和最后的垃圾回收
    void (*class_init)(ObjectClass *klass, void *data);
    void (*class_base_init)(ObjectClass *klass, void *data);
    void (*class_finalize)(ObjectClass *klass, void *data);
    void *class_data;

    // 这个类所实现的接口
    InterfaceInfo *interfaces;
};
```

一个类是如何注册到QOM中的?

QOM提供了type_register和type_register_static方法，用户可以调用这两个方法注册一个Type，需要传进去的参数就是TypeInfo的指针。（代码中）

```
TypeImpl *type_register(const TypeInfo *info)
{
    assert(info->parent);
    return type_register_internal(info);
}

TypeImpl *type_register_static(const TypeInfo *info)
{
    return type_register(info);
}
```

可以看到它们都调用了type_register_internal(TypeInfo*)函数。（代码在qom/object.c中）

```
static TypeImpl *type_register_internal(const TypeInfo *info)
{
    TypeImpl *ti;
    ti = type_new(info);
```

```
    return ti;
}
```

type_new(TypeInfo*)函数将TypeInfo数据结构中的相关信息，赋值给TypeImpl数据结构中的属性。而type_table_add(TypeIm 数将调用t
TypeImpl指针保存到静态的hash表中。
我们回顾一下TypeImpl的数据结构：（代码在qom/object.c中）

```
struct TypeImpl
{
    const char *name;

    size_t class_size;

    size_t instance_size;

    void (*class_init)(ObjectClass *klass, void *data);
    void (*class_base_init)(ObjectClass *klass, void *data);
    void (*class_finalize)(ObjectClass *klass, void *data);

    void *class_data;

    void (*instance_init)(Object *obj);
    void (*instance_post_init)(Object *obj);
    void (*instance_finalize)(Object *obj);

    bool abstract;

    const char *parent;
    TypeImpl *parent_type;

    ObjectClass *class;

    int num_interfaces;
    InterfaceImpl interfaces[MAX_INTERFACES];
};
```

而type_new()函数就是针对TypeInfo中的各种信息，给TypeImpl赋值：（代码在qom/object.c中）

```
static TypeImpl *type_new(const TypeInfo *info)
{
    TypeImpl *ti = g_malloc0(sizeof(*ti));
    int i;

    g_assert(info->name != NULL);

    if (type_table_lookup(info->name) != NULL) {
        fprintf(stderr, "Registering `%s' which already exists\n", info->name);
        abort();
    }

    ti->name = g_strdup(info->name);
    ti->parent = g_strdup(info->parent);

    ti->class_size = info->class_size;
    ti->instance_size = info->instance_size;

    ti->class_init = info->class_init;
    ti->class_base_init = info->class_base_init;
    ti->class_finalize = info->class_finalize;
    ti->class_data = info->class_data;

    ti->instance_init = info->instance_init;
    ti->instance_post_init = info->instance_post_init;
    ti->instance_finalize = info->instance_finalize;

    ti->abstract = info->abstract;

```

```
    }
    ti->num_interfaces = i;

    return ti;
}
```

最后，通过type_table_add()保存到静态的hash表中（代码在qom/object.c中）

```
static void type_table_add(TypeImpl *ti)
{
    assert(!enumerating_types);
    g_hash_table_insert(type_table_get(), (void *)ti->name, ti);
}
static GHashTable *type_table_get(void)
{
    static GHashTable *type_table;  //静态的类型的hash表，保存了全部被注册的类型

    if (type_table == NULL) {
        type_table = g_hash_table_new(g_str_hash, g_str_equal);
    }

    return type_table;
}
```

面向对象特性的实现

封装的实现

在考察QOM如何实现封装时，我们需要再次审视Object这个数据结构包含了哪些属性：（代码在/include/qom/object.h中）

```
struct Object
{
    /*< private >*/
    ObjectClass *class;  //指向对应的类的数据结构的指针
    ObjectFree *free;    //当引用计数为0时调用
    GHashTable *properties;  //Object中的所有属性的hash表
    uint32_t ref;      //对象的引用计数
    Object *parent;    //指向父对象的指针
};
```

值得注意的是，这个Object的数据结构中的一行注释：“private”，它表示Object中的所有属性都是私有的，只能被类的内部函数访问、修改。这
现封装特性的私有变量是properties，它是一张hash表，这个变量包含了Object中的所有可访问和修改的数据、函数。这个Hash表中，每一个ent
value，key是这个property的name，而value是一个ObjectProperty数据结构的指针，ObjectProperty的实现代码如下：（代码在include/qom/

```
typedef struct ObjectProperty
{
    gchar *name;
    gchar *type;
    gchar *description;
    ObjectPropertyAccessor *get;
    ObjectPropertyAccessor *set;
    ObjectPropertyResolve *resolve;
    ObjectPropertyRelease *release;
    void *opaque;
} ObjectProperty;
```

可以看到，ObjectProperty包含了这个属性的名字、类型、描述、get和set的方法，解析（resolve）和释放（release）的方法，以及这个proper
void *类型的指针来表示。
QOM使用这样一个数据结构，将对象的每个数据都保存在这样一个单元之中，从而让实现了封装。
当用户需要向Object中增加属性时，就可以直接调用object_property_add函数，这个函数向object的properties的hash表中插入一个property
qom/object.c中）

```
ObjectProperty *
object_property_add(Object *obj, const char *name, const char *type,
                    ObjectPropertyAccessor *get,
```

```
            void *opaque, Error **errp)
{
    ObjectProperty *prop;
    size_t name_len = strlen(name);

    if (name_len >= 3 && !memcmp(name + name_len - 3, "[*]", 4)) {
        int i;
        ObjectProperty *ret;
        char *name_no_array = g_strdup(name);

        name_no_array[name_len - 3] = '\0';
        for (i = 0; ; ++i) {
            char *full_name = g_strdup_printf("%s[%d]", name_no_array, i);

            ret = object_property_add(obj, full_name, type, get, set,
                                      release, opaque, NULL);
            g_free(full_name);
            if (ret) {
                break;
            }
        }
        g_free(name_no_array);
        return ret;
    }

    if (g_hash_table_lookup(obj->properties, name) != NULL) {
        error_setg(errp, "attempt to add duplicate property '%s'"
                   " to object (type '%s')", name,
                   object_get_typename(obj));
        return NULL;
    }

    prop = g_malloc0(sizeof(*prop));

    prop->name = g_strdup(name);
    prop->type = g_strdup(type);

    prop->get = get;
    prop->set = set;
    prop->release = release;
    prop->opaque = opaque;

    g_hash_table_insert(obj->properties, prop->name, prop);
    return prop;
}
```

继承的实现

继承包括：实现继承、接口继承和可视继承三种。可视继承与图形界面相关，我们不考虑。 实现继承是指使用基类的属性和方法而无需额外编码的
是指仅使用属性和方法的名称、但是子类必须提供实现的能力。

* 实现继承

我们创建一个新类时，需要实现两个数据结构：类的数据结构和对象的数据结构，由于类的数据结构中第一个属性就是父类的数据结构的实例，而
中，第一个属性就是父类对应的对象的实例。这样的实现方式，使得QOM天然地支持显现继承。

* 接口继承

接口在QOM中也有一个对应的类的数据结构：（代码在include/qom/object.h中）

```
struct InterfaceClass
{
    ObjectClass parent_class; //它的父类就是ObjectClass
    /*< private >*/
    ObjectClass *concrete_class;  //实现这个接口的类的指针
    Type interface_type;        //这个interface的类型（TypeImpl*指针），这个属性指示要实现的接口类型。
};
```

在QOM中一个类可以实现多个接口，也就是实现接口继承。在ObjectClass中与接口继承相关的属性就是interfaces，它在ObjectClass是一条链表
entry都是一个InterfaceClass的指针，最终对应到interface的TypeImpl数据结构的指针。我们可以通过给TypeImpl指针对应的类数据结构中的函

```
struct ObjectClass
{
  /*< private >*/
  Type type;
  GSList *interfaces;

  const char *object_cast_cache[OBJECT_CLASS_CAST_CACHE];
  const char *class_cast_cache[OBJECT_CLASS_CAST_CACHE];

  ObjectUnparent *unparent;
};
```

多态的实现

多态是指同一操作作用于不同的对象，可以有不同的解释，产生不同的执行结果。为了实现多态，QOM实现了一个非常重要的功[illegible]是动态cas相关的函数，将一个Object的指针在运行时cast为子类对象的指针，可以将一个ObjectClass的指针在运行时cast为子类的指针。[illegible]可以调用子指针来完成相应的功能。

动态cast检查的主要实现函数是：（代码在qom/object.c中）

```
ObjectClass *object_class_dynamic_cast(ObjectClass *class,
                                       const char *typename)
{
  ObjectClass *ret = NULL;
  TypeImpl *target_type;
  TypeImpl *type;

  if (!class) {
    return NULL;
  }

  type = class->type;  //class的type指针总是指向这个类的TypeImpl，因此从这里获得这个类的类型。
  if (type->name == typename) {
    return class;
  }

  target_type = type_get_by_name(typename);
  if (!target_type) {
    /* target class type unknown, so fail the cast */
    return NULL;
  }
  //检查是否需要动态cast为接口类对象，然后检查每个接口
  //主要检查目标类型是不是当前所指向的类型的祖先。
  if (type->class->interfaces &&
      type_is_ancestor(target_type, type_interface)) {
    int found = 0;
    GSList *i;

    for (i = class->interfaces; i; i = i->next) {
      ObjectClass *target_class = i->data;

      if (type_is_ancestor(target_class->type, target_type)) {
        ret = target_class;
        found++;
      }
    }

    /* The match was ambiguous, don't allow a cast */
    if (found > 1) {
      ret = NULL;
    }
  } else if (type_is_ancestor(type, target_type)) {
    ret = class;
  }

  return ret;
}
```

在Object的数据结构中有一个变量：ref，这个变量用于对Object引用的计数，如果ref的值变为0，就意味着系统不会继续使用这了，那么就的内存空间等进行回收操作。

* 在TypeInfo中，我们可以定义instance_finalize，对于引用计数为0的Object进行垃圾回收操作。
* Object数据结构中有一个ObjectFree *类型的函数指针free，当Object的引用计数为0时，就会调用这个函数进行垃圾回收。
* QOM自己实现了默认的垃圾回收操作：（代码在qom/object.c中）

```
1  //减少obj的引用计数，如果引用计数为0，将进行垃圾回收
2  void object_unref(Object *obj)
3  {
4    if (!obj) {
5      return;
6    }
7    g_assert_cmpint(obj->ref, >, 0);
8
9    /* parent always holds a reference to its children */
10   if (atomic_fetch_dec(&obj->ref) == 1) {
11     object_finalize(obj);
12   }
13 }
14 //obj的默认的析构函数
15 static void object_finalize(void *data)
16 {
17   Object *obj = data;
18   TypeImpl *ti = obj->class->type;
19
20   object_property_del_all(obj); //删除obj中的所有属性
21   object_deinit(obj, ti);  //调用TypeImpl中finalize函数进行析构（请看后面）
22
23   g_assert_cmpint(obj->ref, ==, 0);  //确定引用计数为0
24   if (obj->free) {
25     obj->free(obj);  //如果obj有free函数指针，那么就会调用该函数
26   }
27 }
```

```
1  // 调用TypeImpl中的实例析构函数。如果存在父类，需要继续调用父类的实例析构函数
2  // 调用父类实例析构函数是因为一个对象数据结构中，第一个属性就是父类对象的实例
3  // 当我们需要对对象析构时，不仅要调用当前类的析构方法，也需要调用父类的析构方法
4  // 将对象中的第一个属性进行析构。
5  static void object_deinit(Object *obj, TypeImpl *type)
6  {
7    if (type->instance_finalize) {
8      type->instance_finalize(obj);
9    }
10
11   if (type_has_parent(type)) {
12     object_deinit(obj, type_get_parent(type));
13   }
14 }
```

YuanruiZJU 3年前 欢迎大家提出意见！

QEMU中的对象模型——QOM（介绍篇） 阅 .86

QEMU提供了一套面向对象编程的模型——QOM，即QEMU Object Module，几乎所有的设备如CPU、内存、总线等… 博文 来自： YuanruiZJU

qemu QOM(qemu object model)和设备模拟 阅 769

本文所用qemu为1.5版本的，不是android emulator的。之前几篇文章介绍的都是android emulator中的设备模拟… 博文 来自： ayu_ag的专

QEMU学习笔记——QOM(Qemu Object Model) 阅 190

（转载）本文发自http://www.binss.me/blog/qemu-note-of-qemu-object-model/，转载请注明出处。QOM(Qem… 博文 来自： Hugo的博客

用QEMU和GDB调试Linux内核linux-5.5.9裁剪过的内核配置文件 24

此配置文件.config是针对QEMU模拟器裁剪的linux-5.5.9内核源码配置，以减少编译时间。 https://blog.csdn.net/eidolon… 载

qemu 学习（一）———qemu整体流程解读 阅读数 7659

本文由博主原创，转载请注明出处（保留此处和链接）：IT人生 （http://blog.csdn.net/ningxialieri/article/details… 博文 来自： IT人生

QEMU(1) - QOM 阅读数 287

Table of ContentsTypeInfo根类型TypeInfoTypeInfo链路初始化TypeInfo ->ModuleEntryDump TypeInfoTypeImpl… 博文 来自： lwhuq的博客

qemu代码分析.pdf 01-14

qemu 是使用动态二进制翻译的 cpu 模拟器，它支持两种运行模式：全系统模拟和用户态模拟。在全系统模拟下，qemu … 下载

2-2 type、object和class之间的关系 阅读数 734

慕课网课程：Python高级编程和异步IO并发编程关系图解释如下：1.type是一个对象，它是type本身的一个实例pri… 博文 来自： shfscut的博客

qemu2的qom系统分析(-)对象系统 阅读数 85

前边分析machine的注册和选择，发现如果 不了解qom系统是很难分析的。qom系统的说明在include/qom/object.… 博文 来自： woai110120130的…

QEMU中的对象模型——QOM(介绍篇)_YuanruiZJU的博客-CSDN博客

QEMU中协程的实现和使用_YuanruiZJU的博客-CSDN博客

1.1Qemu 用户态架构 阅读数 4520

本节首先分析Qemu的初始化的顶层流程;从而引出Qemu各大功能模块的描述；最后分析Qemu与内核态KVM的通讯… 博文 来自： wanthelping的博客

QEMU学习笔记——QOM(Qemu Object Model) - Hugo的博客 - CSDN博客

QEMU中的内存管理介绍_YuanruiZJU的博客-CSDN博客

Java 中 类 对象 实例 类的对象 对象的实例 对象的引用 概念-- 阅读数 1万+

Java中类对象实例类的对象对象的实例对象的引用概念 博文 来自： 逐梦笔记

QEMU中的CPU类型设计_YuanruiZJU的博客-CSDN博客

QEMU 设备模拟 阅…… …477

设备模拟目的我们好像不会干一件事而毫无目的，就算不停刷微信朋友圈也是为了打发你无聊的时间。其实最装B的… 博文 来自： 万能的终端…

QEMU中如何定义所有Device的基类和BUS的基类 阅 …179

本文介绍QEMU如何模拟设备、总线、主板的连接关系。 博文 来自： YuanruiZJU…

...定义所有Device的基类和BUS的基类_YuanruiZJU的博客-CSDN博客

qemu QOM(qemu object model)和设备模拟_ayu_ag的专栏-CSDN博客

QEMU介绍 阅 …'55

QEMU 作为开源界最热门的 Emulator，被广泛的应用在诸如内核调试，跨平台开发等领域。例如现在很火的 Androi… 博文 来自： pkufergus的…

Qemu中coroutine机制的实现 阅读数 2637

最近在看virtio的代码，看到virtio后端时发现在Qemu处理IO的时候使用了coroutine，之前对coroutine不了解，因… 博文 来自： 慢慢游

开源KVM管理平台 阅读数 6643

管理平台的选择问题，在其他虚拟化引擎上不是问题，但是在KVM上却是百花齐放，没有一个管理平台能够拿来直接… 博文 来自： Enweitech Softwa…

QEMU虚拟机源码分析概论 阅读数 4683

QEMU官网描述QEMU的官网首页上对其自身的描述如下（请特别注意标红字的部分）：QEMU is a generic and ope… 博文 来自： YuanruiZJU的博客

KVM虚拟机代码揭秘——QEMU代码结构分析 阅读数 2万+

如我们所知，QEMU是一个模拟器，它能够动态模拟特定架构的CPU指令，如X86，PPC，ARM等等。QEMU模拟的… 博文 来自： Shawn的专栏

QEMU 简单几步搭建一个虚拟的ARM开发板 阅读数 4792

1.安装QEMU先在Ubuntu中安装QEMUsudoapt-getinstallqemu安装几个QEMU需要的软件包：sudoapt-getinstallz… 博文 来自： leumber的专栏

Qemu使用心得 阅读数 460

使用Qemu的心得体会如下：（1）在QEMU源码中增加自己的.c实现，编译后出现很多个错误如：“error: storage c… 博文 来自： lyw736632087的…

QEMU中的内存管理介绍 阅读数 4603

@(qemu源码阅读与分析)QEMU中的内存管理(介绍)QEMU在虚拟机启动的初始化阶段，为客户机分配了物理内存，… 博文 来自： YuanruiZJU的博客

HBitmap、Bitmap、CBitmap之间的区别与联系 阅读数 4958

一、术语的定义及含义：HANDLE： 是DIB的文件句柄,是表示了设备无关位图在内存中的区域存储代号,该句柄值… 博文 来自： self_mind的记事本

qemu中的Hbitmap数据结构 阅读数 714

尽管unsigned long已经能够胜任bitmap的实现，但是这样实现的bitmap，当bitmap比较大的时候，它的操作效率… 博文 来自： YuanruiZJU的博客

qemu HBitmap原理 阅 …34

一：磁盘位图HBitmap分析使用unsigned long已经能够胜任bitmap的实现，但是当bitmap比较大的时候，它的操… 博文 来自： weixin_34…

大学四年自学走来，这些私藏的实用工具/学习网站我贡献出来了
大学四年，看课本是不可能一直看课本的了，对于学习，特别是自学，善于搜索网上的一些资源来辅助，还是非常有… 博文 来自：帅地

在中国程序员是青春饭吗？
今年，我也32了 ，为了不给大家误导，咨询了猎头、圈内好友，以及年过35岁的几位老程序员……舍了老脸去揭人… 博文 来自：启舰

超全Python图像处理讲解（多图预警）
文章目录Pillow模块讲解一、Image模块1.1 、打开图片和显示图片1.2、创建一个简单的图像1.3、图像混合（1）透… 博文 来自：ZackSock的

为什么猝死的都是程序员，基本上不见产品经理猝死呢？
相信大家时不时听到程序员猝死的消息，但是基本上听不到产品经理猝死的消息，这是为什么呢？我们先百度搜一下… 博文 来自：曹银飞的专

毕业5年，我问遍了身边的大佬，总结了他们的学习方法
我问了身边10个大佬，总结了他们的学习方法，原来成功都是有迹可循的。 博文 来自：敖丙

推荐10个堪称神器的学习网站 阅读数 26万+
每天都会收到很多读者的私信，问我：“二哥，有什么推荐的学习网站吗？最近很浮躁，手头的一些网站都看烦了，… 博文 来自：沉默王二

Java校招入职华为，半年后我跑路了 阅读数 19万+
何来我，一个双非本科弟弟，有幸在 19 届的秋招中得到前东家华为（以下简称 hw）的赏识，当时秋招签订就业协议… 博文 来自：JavaEdge

强烈推荐10本程序员必读的书 阅读数 8万+
很遗憾，这个春节注定是刻骨铭心的，新型冠状病毒让每个人的神经都是紧绷的。那些处在武汉的白衣天使们，尤其… 博文 来自：沉默王二

为什么说程序员做外包没前途？ 阅读数 10万+
之前做过不到3个月的外包，2020的第一天就被释放了，2019年还剩1天，我从外包公司离职了。我就谈谈我个人的… 博文 来自：dotNet全栈开发

B 站上有哪些很好的学习资源？ 阅读数 13万+
哇说起B站，在小九眼里就是宝藏般的存在，放年假宅在家时一天刷6、7个小时不在话下，更别提今年的跨年晚会，… 博文 来自：九章算法的博客

敲完1万行代码后，我终于找到了程序员快速入门的办法
程序员太难了

昂，我24岁了 阅读数 3万+
24岁的程序员，还在未来迷茫，不知道能不能买得起房子 博文 来自：敖丙

新来个技术总监，禁止我们使用Lombok！ 阅读数 3万+
我有个学弟，在一家小型互联网公司做Java后端开发，最近他们公司新来了一个技术总监，这位技术总监对技术细节… 博文 来自：HollisChuang's Blog

字节跳动的技术架构 阅读数 2万+
字节跳动创立于2012年3月，到目前仅4年时间。从十几个工程师开始研发，到上百人，再到200余人。产品线由内涵… 博文 来自：作一个独立连续的…

在三线城市工作爽吗？ 阅读数 8万+
我是一名程序员，从正值青春年华的 24 岁回到三线城市洛阳工作，至今已经 6 年有余。一不小心又暴露了自己的实… 博文 来自：沉默王二

这些插件太强了，Chrome 必装！尤其程序员！
推荐 10 款我自己珍藏的 Chrome 浏览器插件 博文 来自：沉默王二

学Python的程序员建议收藏！

YuanruiZJU
TA的个人主页 >

原创	粉丝	获赞	评论	访问
15	28	7	5	3万+

等级: 博客 3 周排名: 19万+
积分: 563 总排名: 14万+
勋章:

关注 私信

热门文章

最新评论