JVM启动流程：https://www.processon.com/view/link/602f65355653bb64f21d7453
线程创建流程：https://www.processon.com/view/link/613dbb621efad46d32e604bc
STW代码：https://gitee.com/luban-ziya/ziya-stw-cpp

有道云链接：http://note.youdao.com/noteshare?id=d17b07353b7bda12ab184588ba0e2646&sub=1BA5CFF481304D36B3516DD6EA0D716C

【2.16】一、Hotspot初探：用Java实现JVM框架
【2.18】二、Hotspot初探：深入理解内存模型与GC
【2.20】三、Hotspot初探：深入理解多线程
三、Hotspot初探：深入理解多线程
polling_page
STW底层实现原理
如何手写实现STW（源码明天给大家）
JVM底层是如何实现线程的
成为一个技术大牛一定是没错的
教学
自身理解
学习视角 第三方
设计者的角度
1、知道与不知道
2、懂与不懂
3、做与不做
Linux
OS内核
我欠缺什么
new Thread
8
200
打开文件 190
crash
1、特性：是否自动释放资源
2、业务场景
fd
socket
运行完自己回收

1、如何才能手写线程池、synchronized
2、Linux多线程机制
如何创建线程
线程类型
默认 joinable
分离 detached
如何阻塞线程
如何唤醒线程
如何遍历线程
pthread_create
线程类型
默认 joinable
构成父子线程管理
占用的资源
分离 detached
脱离管控
失控
这个线程运行结束，占用的资源也会马上回收
描述符 fd socket
crash
open
socket
线程返回值
1、线程类型
默认 joinable
分离 detached
2、如何获取线程的返回值
joinable
两种方式
detached
如何获取
Thread vm_result vm_result2
注意控制线程顺序
无序
互斥
同步
无序
基本类型

引用类型
设置线程属性
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
互斥锁
pthread_mutex_init
pthread_mutex_lock
pthread_mutex_unlock
条件标量
pthread_cond_init
pthread_cond_wait
pthread_cond_signal 唤醒单个线程
pthread_cond_broadcast 唤醒所有线程
park
阻塞线程
底层实现：互斥锁+条件标量
pthread_mutex_lock
pthread_cond_wait
pthread_mutex_unlock
线程池
如果任务池中没有任务 所有线程阻塞
向任务池中添加了任务，唤醒所有线程
unpark
唤醒线程
底层实现：互斥锁+条件标量
pthread_mutex_lock
pthread_cond_signal\pthread_cond_broadcast
pthread_mutex_unlock
看代码
3、Linux信号处理机制
signal
kill -9
发生了除零异常
java代码除零异常
JVM注册了信号处理机制
JVM定义信号处理函数
JVM是怎么知道的

段异常
JVM的安全点就是这样实现的
4、STW底层实现原理
信号机制
暂停所有用户线程
1、如何找到所有的线程
Linux没有提供这种api
你得自己做工作
你得搞一个容器存储
遍历
单链表
2、线程是如何知晓安全点已被激活
触发了段异常
段异常是如何触发的
内存 可读可写的属性的
test %eax, os::_polling_page;
可读的
不可读
3、安全点是什么
一块特殊的内存
_polling_page
hsdis
test %eax, os::_polling_page;
1、这个内存页是如何创建的
2、如何设置为不可读
make_polling_page_unreadable
3、如何设置为可读
make_polling_page_readable
4、注册信号处理
4、安全点插在哪里
想达到的效果
让线程快速进入安全点
程序有可能长时间运行的位置都要插安全点
hsdis
STW
GC
5、带你手写实现STW
代码明天给大家

6、读Hotspot源码深入理解Java线程
new Thread().start();
new Thread()
就创建了一个java对象，JVM层面就是一个oop，其他什么事都没干
start
真正创建线程的逻辑全部在这一步
native_thread = new JavaThread(&thread_entry, sz);
继承自Thread
JavaThread::JavaThread
set_entry_point(entry_point);
os::create_thread(this, thr_type, stack_sz);
OSThread* osthread = new OSThread(NULL, NULL);
pthread_create(&tid, &attr, (void* (*)(void*)) java_start,
// notify parent thread
osthread->set_state(INITIALIZED);
sync->notify_all();
// wait until os::start_thread()
wait
this->entry_point()(this, this);
osthread->set_pthread_id(tid);
Wait until child thread is either initialized or aborted
Thread::Thread 基本上没什么有价值的信息
native_thread->prepare(jthread);
Thread::start(native_thread);
notify 子线程
java的线程创建至少分两步：创建后阻塞、准备工作做完唤醒执行
提问题
找答案
四者之间的关联代码
new Thread这个java对象
JavaThread
java_lang_Thread::set_thread(thread_oop(), this);
OSThread c++对象
thread->set_osthread(osthread);
操作系统线程
tid
osthread->set_pthread_id(tid);
// Wait until child thread is either initialized or aborted

run方法是如何运行的
执行流，在内存中
call_stub
entry_point
画栈图、堆栈图
不画就看不懂
javaCall 奈何桥
java世界 | JVM世界
static void thread_entry(JavaThread* thread, TRAPS) {
HandleMark hm(THREAD);
Handle obj(THREAD, thread->threadObj());
JavaValue result(T_VOID);
JavaCalls::call_virtual(&result,
obj,
KlassHandle(THREAD, SystemDictionary::Thread_klass()),
vmSymbols::run_method_name(),
vmSymbols::void_method_signature(),
THREAD);
}
JavaCalls::call_helper
// function pointer c语言
StubRoutines::call_stub()(
(address)&link,
// (intptr_t*)&(result->_value), // see NOTE above (compiler problem)
result_val_address, // see NOTE above (compiler problem)
result_type,
method(),
entry_point,
args->parameters(),
args->size_of_parameters(),
CHECK
);
1、看揭秘那本书
2、堆栈图（汇编）
3、自实现call_stub（我写了）
线程阻塞
主线程在启动子线程的时候，主线程需要阻塞