📭 sdk ⟩ 🖿 runtime ⟩ 🖿 lib ⟩ 📇 isolate.cc

#### 1、Isolate创建:

```
sdk > sdk > isolate_patch.dart
```

### 2、Isolate.spawn 的内部实现会调用\_spawnFunction函数

```
⊕ 🚡 🕏 − 🚜 isolate_patch.dart × 📠 isolate.cc × 🚓 thread_pool.cc × 🚓 thread_pool.cc
thread_barrier.h
thread_barrier_test.cc
                                                                                                                                                               DISALLOW_COPY_AND_ASSIGN(SpawnIsolateTask);
                                                                                                                                                            static const char* String2UTF8(const String& str) {
  intptr_t len = Utf8::Length(str);
  char* result = new char[len + 1];
  str.ToUTF8(reinterpret_cast<uint8_t*>(result), len);
  result[len] = 0;
  thread_interrupter_fuchsia.cc
   thread_interrupter_linux.cc
  # thread_interrupter_macos.cc
  thread_pool.cc
                                                                                                                                                        DEFINE_NATIVE_ENTRY(Isolate_spawnFunction, 0, 11) {
    GET_NON_NULL_NATIVE_ARGUMENT(SendPort, port, arguments->NativeArgAt(0));
    GET_NON_NULL_NATIVE_ARGUMENT(String, script_uri, arguments->NativeArgAt(1));
    GET_NON_NULL_NATIVE_ARGUMENT(Instance, closure, arguments->NativeArgAt(2));
    GET_NON_NULL_NATIVE_ARGUMENT(Instance, message, arguments->NativeArgAt(3));
    GET_NON_NULL_NATIVE_ARGUMENT(Bool, paused, arguments->NativeArgAt(4));
    GET_NATIVE_ARGUMENT(Bool, fatalErrors, arguments->NativeArgAt(5));
    GET_NATIVE_ARGUMENT(SendPort, onExit, arguments->NativeArgAt(6));
    GET_NATIVE_ARGUMENT(SendPort, onError, arguments->NativeArgAt(7));
    GET_NATIVE_ARGUMENT(String, packageRoot, arguments->NativeArgAt(8));
    GET_NATIVE_ARGUMENT(String, packageConfig, arguments->NativeArgAt(9));
    GET_NATIVE_ARGUMENT(String, debugName, arguments->NativeArgAt(10));
  thread_pool_test.cc
  thread_registry.h
thread_stack_resource.cc
  thread_stack_resource.h
thread_state.cc
  # thread_test.cc
  timeline.h
                                                                                                                                                        if (closure.Isclosure()) {
   Function& func = Function::Handle();
   func = Closure::Cast(closure).function();
   if (func.IsImplicitClosureFunction() && func.is_static()) {
   #if defined(DEBUG)
   Context& ctx = Context::Handle();
   ctx = Closure::Cast(closure).context();
   ASSERT(ctx.IsNull());
   #endif
  timeline_android.cc
   atimeline_fuchsia.cc
  atimer.cc
                                                                                                                                                                         // Get the parent function so that we get the right function name
func = func.parent_function();
   # token.cc
```

```
sdk [~/Desktop/sdk] - .../runtime/vm/thread_pool.h [sdk]
sdk > Im runtime > Im vm > thread_pool.h
                                                                         athread_barrier.h
                # thread_barrier_test.cc
                thread_interrupter.cc
                                                                                          class ThreadPool {
                # thread_interrupter.h
                                                                                            // Subclasses of Task are able to run on a ThreadPool.
class Task {
                thread_interrupter_android.cc
                athread_interrupter_fuchsia.cc
                                                                                              protected:
  Task();
                ## thread_interrupter_linux.cc
                 thread_interrupter_win.cc
                athread_pool.cc
                                                                                               // Override this to provide task-specific behavior.
virtual void Run() = 0;
                dthread_pool.h
                ## thread_pool_test.cc
                thread_registry.cc
                thread_registry.h
thread_stack_resource.cc
                                                                                               DISALLOW_COPY_AND_ASSIGN(Task);
                thread_stack_resource.h
                                                                                             ThreadPool();
                # thread_state.cc
                                                                                            // Shuts down this thread pool. Causes workers to terminate // themselves when they are active again. 
 \sim\!\! \text{ThreadPool()};
                athread_state.h
                thread_test.cc
                timeline.cc
                                                                                            // Runs a task on the thread pool.
template <typename T, typename... Args>
bool Run(Args&&... args) {
   return RunImpl(std::unique_ptr<Task>(new T(std::forward<Args>(args)...)));
                 analysis.cc
                 analysis.h
                 android.cc timeline_android.cc
                timeline_fuchsia.cc
                                                                                            // Some simple stats.
uint64_t workers_running() const { return count_running_; }
uint64_t workers_idle() const { return count_idle_; }
uint64_t workers_started() const { return count_started_; }
uint64_t workers_stopped() const { return count_stopped_; }
                timeline_test.cc
                timer.h
```

### 3、

```
sdk [~/Desktop/sdk] - .../runtime/vm/thread_pool.cc [sdk]
sdk ⟩ I runtime ⟩ I vm ⟩ a thread_pool.cc
                                                                                                                                                                          🖟 🔲 | Git: 🗹 🗸 🕓 🖒 📭 🖪
                                                    ⊕ 😤 🌣 — 🚜 isolate_patch.dart × 🚉 isolate.cc × 🚉 thread_pool.cc × 🚉 thread_pool.cc
               thread_barrier.h
               thread_barrier_test.cc
                                                                                    bool ThreadPool::RumImpl(std::unique_ptr<Task> task) {
  Worker* worker = NULL;
               thread_interrupter.cc
               thread_interrupter_android.cc
               athread_interrupter_fuchsia.cc
                # thread interrupter linux.cc
                                                                                         MutexLocker ml(&mutex_);
if (shutting_down_) {
                # thread interrupter macos.cc
               # thread_interrupter_win.cc
                                                                                          if (idle_workers_ == NULL) {
  worker = new Worker(this);
  ASSERT(worker != NULL);
               thread_pool.h
               thread_pool_test.cc
               thread_registry.cc
                                                                                            count started ++
                athread_registry.h
                                                                                            // Add worker to the all_workers_ list.
worker->all_next_ = all_workers_;
                                                                                           all_workers_ = worker;
worker->owned_ = true;
count_running_++;
               athread_state.cc
               athread_test.cc
                                                                                           else {
// Get the first worker from the idle worker list.
worker = idle_workers_;
idle_workers_ = worker->idle_next_;
worker->idle_next_ = NULL;
               atimeline.cc
                atimeline.h
                timeline_analysis.cc
                                                                                           count_running_++;
               timeline_analysis.h
               android.cc
                                                                                       // Release ThreadPool::mutex_ before calling Worker functions.
ASSERT(worker != NULL);
               timeline_test.cc
                timer.cc
                                                                                       worker->SetTask(std::move(task));
               timer.h
                                                                                        worker>dif (new_worker) {
    // Call StartThread after we've assigned the first task.
    worker->StartThread();
               d token.h
               token_position.cc
                token_position.h
                atype_table.h
```

## 4、

```
sdk > runtime > m vm > thread_pool.cc
                                                ⊕ 🚡 🌣 — 🕻 isolate_patch.dart × 📇 isolate.cc × 📇 thread_pool.cc × 📇 thread_pool.h
             thread_barrier.h
                                                                                      owned_(false),
all_next_(NULL),
idle_next_(NULL),
shutdown_next_(NULL) {}
             thread_interrupter.cc
             d thread_interrupter_android.cc
             thread_interrupter_fuchsia.cc
                                                                                ThreadId ThreadPool::Worker::id() {
  MonitorLocker ml(&monitor_);
             thread_interrupter_linux.cc
             athread_interrupter_win.cc
            thread_pool.cc
                                                                               void ThreadPool::Worker::StartThread() {
#if defined(DEBUG)
// Must call SetTask before StartThread.
                                                                                 // Must call Sectask Defore 5
{    // NOLINT
    MonitorLocker ml(&monitor_);
    ASSERT(task_ != nullptr);
             ## thread_registry.cc
             thread_registry.h
             thread_stack_resource.cc
             thread_stack_resource.h
                                                                                 thread_state.cc
             thread_state.h
thread_test.cc
                                                                                  if (result != 0) {
FATAL1("Could not start worker thread: result = %d.", result);
             timeline.cc
```

## 5、创建线程

```
runtime m vm ≥ cos_thread_macos.cc
                                                                       🔲 <no devices> 🔻 Add Configuration... 🖟 Nexus 5X API 27 🔻 👂 😭 🚍 🎂 🕠 🕢 🖟 🔲 🖸 Git: 🖌 🗸 🕓 🐚
                                                    🛟 🚡 🕏 — 🕻 isolate_patch.dart × 👛 isolate.cc × 👛 thread_pool.cc × 👛 os_thread_macos.cc × 👛 thread_pool.h
                                                                                       OSThread* thread = OSThread::CreateOSThread();
if (thread != NULL) {
              thread_barrier.h
               ## thread_barrier_test.cc
                                                                                         f (thread != NULL) t
  OSThread::SetCurrent(thread);
  thread->set_name(name);
  // Call the supplied thread start function handing it its parameters.
               # thread_interrupter.cc
               thread_interrupter.h
                                                                                          function(parameter);
               # thread_interrupter_fuchsia.cc
               ## thread_interrupter_linux.cc
                                                                                       return NULL;
               ## thread_interrupter_macos.cc
               thread_interrupter_win.cc
                                                                                     athread_pool.cc
               thread_pool.h
              dthread_pool_test.cc
                                                                                       pthread_attr_t attr;
int result = pthread_attr_init(&attr);
RETURN_ON_PTHREAD_FAILURE(result);
               thread_registry.h
               athread_stack_resource.cc
                                                                                       "result = pthread_attr_setstacksize(&attr, OSThread::GetMaxStackSize());
RETURN_ON_PTHREAD_FAILURE(result);
               thread_stack_resource.h
               ## thread_state.cc
                                                                                       ThreadStartData* data = new ThreadStartData(name, function, parameter);
               thread_state.h
               ## thread_test.cc
                                                                                       pthread_t tid;
                                                                                       result = pthread_create(&tid, &attr, ThreadStart, data);
RETURN_ON_PTHREAD_FAILURE(result);
               atimeline.cc
               atimeline.h
               timeline_analysis.cc
                                                                                       result = pthread_attr_destroy(&attr);
RETURN_ON_PTHREAD_FAILURE(result);
               # timeline_analysis.h
               # timeline_android.cc
               atimeline_fuchsia.cc
               timeline_linux.cc
                                                                  □ <no devices> ▼ | Add Configuration... □ Nexus 5X API 27 ▼ ▶ ☆ 등 券 □ ∳ □ Git: ✔ ✔ ⊙ 与 ▶ 📻 🗖 📮
ਫ sdk 〉 I runtime 〉 I vm 〉 a thread_pool.cc
                                                 ⊕ 🚡 🌣 — 🕻 isolate_patch.dart × 📇 isolate.cc × 🚉 thread_pool.cc × 📥 thread_pool.h
             athread_barrier.h
                                                                                                           \uparrow \downarrow \Omega \uparrow \uparrow \Pi \square \square \square \square \square \square Match Case \square Words \square Regex ? 2 matches
              ## thread_barrier_test.cc
                                                                                   UNREACHABLE();
              athread_interrupter.h
                                                                                void ThreadPool::Worker::Shutdown() {
  MonitorLocker ml(&monitor_);
              thread_interrupter_linux.cc
                                                                                   done_ = true;
ml.Notify();
              athread_interrupter_win.cc
              thread_pool.cc
                                                                                // static
void ThreadPool::Worker::Main(uword args) {
  Worker* worker = reinterpret_cast<Worker*>(args);
  OSThread* os_thread = OSThread::Current();
  ASSERT(os_thread != NULL);
  ThreadId id = os_thread->id();
  ThreadPool* pool:
              thread_pool.h
              thread_pool_test.cc
             thread_stack_resource.h
thread_state.cc
                                                                                  {
MonitorLocker ml(&worker->monitor_);
ASSERT(worker->task_);
worker->id_ = id;
worker->id_ = pocker->pool;
              thread_test.cc
```

# 6. Worker —> Loop

timeline\_fuchsia.cc

```
sdk ⟩ I runtime ⟩ I vm ⟩ a thread_pool.cc
                                                                                                                                                                                  ⊕ 🚡 🌣 — 🚜 isolate_patch.dart × 📇 isolate.cc × 📇 thread_pool.cc × 😅 os_thread_macos.cc × 📇 thread_pool.h
                                                  thread_barrier.h
                                                   thread_barrier_test.cc
                                                 thread_interrupter.cc
                                                                                                                                                                                                                                                                                                  bool ThreadPool::Worker::Loop() {
  MonitorLocker ml(&monitor_);
                                                   athread_interrupter.h
                                                  thread_interrupter_android.cc
                                                                                                                                                                                                                                                                                                          int64_t idle_start;
                                                   athread_interrupter_fuchsia.cc
                                                                                                                                                                                                                                                                                                                  hile (true) {
   ASSERT(task_ != nullptr);
                                                   thread_interrupter_linux.cc
                                                                                                                                                                                                                                                                                                                    std::unique_ptr<Task> task = std::move(task_);
                                                   thread_interrupter_macos.cc
                                                                                                                                                                                                                                                                                                                    // Release monitor while handling the task.
                                                  thread_interrupter_win.cc
                                                                                                                                                                                                                                                                                                                 ml.Exit();
                                                  d thread_pool.cc
                                                                                                                                                                                                                                                                                                                  std::atomic_thread_fence(std::memory_order_acquire);
                                                  athread_pool.h
                                                   thread_pool_test.cc
                                                  athread_registry.cc
                                                                                                                                                                                                                                                                                                                 ml.Enter():
                                                   thread_registry.h
                                                   thread_stack_resource.cc
                                                                                                                                                                                                                                                                                                                 ASSERT(task_ == nullptr);
if (IsDone()) {
  return false;
                                                   thread_stack_resource.h
                                                   athread_state.h
                                                                                                                                                                                                                                                                                                                //
ASSERT(!done_);
pool_~SetIdleAndReapExited(this);
idle_start = OS::GetCurrentMonotonicMicros();
while (true) {
    Monitor::WaitResult result = ml.WaitMicros(ComputeTimeout(idle_start));
    idle_start = Ml.WaitMicros(ComputeTimeout(idle_start));

destallation destallation destallation design 
                                                  atimeline.cc
                                                  atimeline.h
                                                   timeline_analysis.cc
                                                                                                                                                                                                                                                                                                                          if (task_ != nullptr) {
   // We've found a task. Process it, regardless of whether the
   // worker is done_.
   break;
                                                   analysis.h
                                                   dimeline_android.cc
                                                  atimeline_fuchsia.cc
                                                                                                                                                                                                                                                                                                                        }
if (IsDone()) {
    return false;
                                                   timeline_linux.cc
                                                  atimeline_test.cc
                                                 atimer.cc
                                                timer.h
                                                                                                                                                                                                                                                                                                                           if ((result == Monitor::kTimedOut) && pool_->ReleaseIdleWorker(this)) {

de token.h

d
                                                   de token_position.cc
                                                                                                                                                                                                                                                                                                          UNREACHABLE();
                                                   atoken_position.h
                                                   type_table.h
```

// It should be okay to access these unlocked here in this assert.
// worker->all\_next\_ is retained by the pool for shutdown monitoring.
ASSERT(!worker->owned\_ && (worker->idle\_next\_ == NULL));

# 7、SpawnIsolateTask -> Run

```
sdk ⟩ Im runtime ⟩ Im lib ⟩ 🚓 isolate.cc
                                             ⊕ 🚡 🌣 - 🖟 isolate_patch.dart × 📠 isolate.cc × 👛 thread_pool.cc × 👛 os_thread_macos.cc × 👛 thread_pool.h
                                                                           SpawnIsolateTask(Isolate*, parent_isolate, std::unique_ptr<IsolateSpawnState> state)
: parent_isolate_(parent_isolate), state_(std::move(state)) {
    parent_isolate->IncrementSpawnCount();
             athread_barrier.h
             ## thread_barrier_test.cc
             thread_interrupter.cc
             thread_interrupter_android.cc
                                                                           ~SpawnIsolateTask() override {
   if (parent_isolate_ != nullptr) {
     parent_isolate_->DecrementSpawnCount();
}
             athread_interrupter_fuchsia.cc
             thread_interrupter_macos.cc
thread_interrupter_win.cc
             athread_pool.cc
                                                                           void Run() override {
  auto group = state_->isolate_group();
             thread_pool.h
                                                                             // The create isolate group call back is mandatory. If not provided we \/\/ cannot spawn isolates.
                                                                             dhread_registry.h
             thread_stack_resource.cc
             ## thread_stack_resource.h
## thread_state.cc
### thread_state.h
             athread_test.cc
             atimeline.cc
            timeline.h
                                                                             analysis.h
                                                                             android.cc
             timeline_linux.cc
```