分布式系统作业-1

18340166 王若琪

---

1. 分布式系统的扩展方式有哪些？各有哪些利弊?

- **隐藏通信延迟**，尽量避免等待远程服务队请求的响应，可以通过利用异步通信技术、设计分离的响应消息处理器（分布）来实现。
  - 利：基本想法很简单，尽量避免等待远程服务队请求的响应，比如在发出请求后，可以利用这个时间做其他工作，这样能够加快运行速度，使系统运行效率更高。
  - 弊：并不是所有应用都适合这种模式，比如依赖上一步结果的应用不能异步。某些交互式应用的用户发出请求后，处于等待无所事事的状态。
- **分布技术**，在多个机器上划分数据和计算，分布技术把某个组件分割成多个部分，然后再将它们分散到系统中去。
  - 利：避免了单个服务器处理大量数据的困境，将任务分到不同计算机上，减轻机器负担。
  - 弊：难以实现负载平衡，有时会出现性能下降问题。
- **复制**，**复制和缓存**：在多个不同的机器上创建多个数据副本。
  - 利：使用简单，增强可靠性，有助于实现负载平衡。
  - 弊：存在一致性问题。有多个副本时，修改一个副本后会导致该副本与其他内容不一致；总是保证副本的一致性需要对每次修改进行全局同步；全局同步代价高昂，难以应用到大规模的解决方案中。

---

2. 分布式系统设计面临哪些挑战？请举例回答。

- **系统设计**
  - 正确的接口设计和抽象；
  - 如何换分功能和可扩展性；
- **一致性**
  - 如何一致性共享数据；
- **容错**
  - 如何保障系统在出现失效情况下正常运行；
- **不同的部署场景**
  - 集群；
  - 广域分布；
  - 传感网络；
- **实现**
  - 如何最大化并行；
  - 性能的瓶颈是什么；
  - 如何平衡负载；

---

3. 请从一些开源分布式软件中找出能够体现透明性的样例代码，并解释是何种类型的透明性。

- Pomegranate 是一个数据库创建和运行模式迁移的工具，它强调安全性和透明性。如果有一个迁移引发异常，事件将自动 roll back，确保数据库不会处于半迁移状态。

迁移透明性是指分布式系统中的资源移动不会影响该资源的访问方式，故障透明性是指用户不会注意到系统自动处理故障的过程。这个工具的代码体现了迁移透明性和故障透明性。

下面这段代码摘自 https://github.com/btubbs/pomegranate ，是有关迁移过程中如果出现差错就自动 roll back 的一小部分，这段关于故障处理的代码，体现了 **迁移透明性** 和 **故障透明性** ：

```
// MigrateBackwardTo will run backward migrations starting with the most recent
// in state, and going through the one provided in `name`.
func MigrateBackwardTo(name string, db *sql.DB, allMigrations []Migration, confirm bool) error {
    if len(allMigrations) == 0 {
        return errors.New("no migrations provided")
    }
    state, err := GetMigrationState(db)
    if err != nil {
        return fmt.Errorf("could not get migration state: %v", err)
    }
    // if nothing in state, nothing to do. error
    if len(state) == 0 {
        return errors.New("state is empty. cannot migrate back")
    }
    toRun, err := getMigrationsToReverse(name, state, allMigrations)
    if err != nil {
        return err
    }
    // get confirmation on the list of backward migrations we're going to run
    if confirm {
        if err := getConfirm(toRun, "Backward", os.Stdin); err != nil {
            return err
        }
    }
    // run the migrations
    for _, mig := range toRun {
        err = runMigrationSQL(db, mig.Name, mig.BackwardSQL)
        if err != nil {
            return err
        }
    }
    return nil
}
//  本段代码摘自https://github.com/btubbs/pomegranate
```

- 下面这段代码体现了 **重定位透明性** 。它摘自 https://github.com/MeteorCode/LavaScript ， 是有关于分布式的 JavaScript 环境的重定位透明性，重定位透明性是指隐藏资源是否在使用过程中被移动到另一个位置，在如下这段代码中，执行引擎被定义为一个私有类，这样就可以保证它不会被修改到外部，这样就不需要用户每次都重新注入绑定。

```
package com.meteorcode.lavascript
import org.dynjs.runtime.DynJS
import org.dynjs.jsr223.{DynJSScriptEngineFactory, DynJSScriptEngine}
import javax.script.{ScriptEngine, ScriptEngineFactory, Bindings, ScriptContext}
import scala.collection.Map
```

```
import scala.collection.JavaConverters.{ mapAsJavaMapConverter
                                       , mapAsScalaMapConverter }
import scala.language.postfixOps
import scala.util.Try

class State( bindings: Map[String, Object]
           , factory: ScriptEngineFactory = new
DynJSScriptEngineFactory )
extends StateLike[State] {
  private[this] val e: Executor = Executor(factory.getScriptEngine)
  e.bindings() putAll bindings.asJava
  override def +=(kv: (String, Object))
    = { e.bindings()
         .put(kv._1, kv._2.asInstanceOf[Object])
        this
      }
  override def -=(key: String)
    = { e.bindings() remove key
        this
      }
  override def get(key: String): Option[Object]
    = Option(e.bindings() get key)
  override def iterator: Iterator[(String, Object)]
    = e.bindings().asScala.iterator
  override def bind(script: String): Try[State]
    = Try (e eval script ) map (_ => this)
  def invoke(function: String, args: AnyRef*): Try[State]
    = ??? // it would be nice to wrap the DynJs invoke method
function...
  /** Wrapper class for a State's JS execution engine.
    *
    * The execution engine is defined as a private inner class so that
we can
    * swear it isn't modified externally. This saves us from having to
    * re-inject bindings every time, as we can prove nobody else has
touched
    * the JavaScript universe.
    */
  case class Executor(engine: ScriptEngine) {
    @inline def bindings(): java.util.Map[String, Object]
      = engine.getBindings(ScriptContext.GLOBAL_SCOPE)
    @inline def eval(script: String): Try[Object]
      = Try(engine eval(script,
engine.getBindings(ScriptContext.GLOBAL_SCOPE)))
  }
}
 /*本段代码摘自 https://github.com/MeteorCode/LavaScript */
```

- **访问透明性** 是指隐藏数据表示形式的不同以及资源访问方式的不同。我选取的这些代码主要是有关响应分布的服务，功能是将 cli 协作带到云端，并完善了服务器的访问透明性。由于代码过长不方便粘贴，所以将链接复制下来以供参考：

  https://github.com/daixtr/mewesh