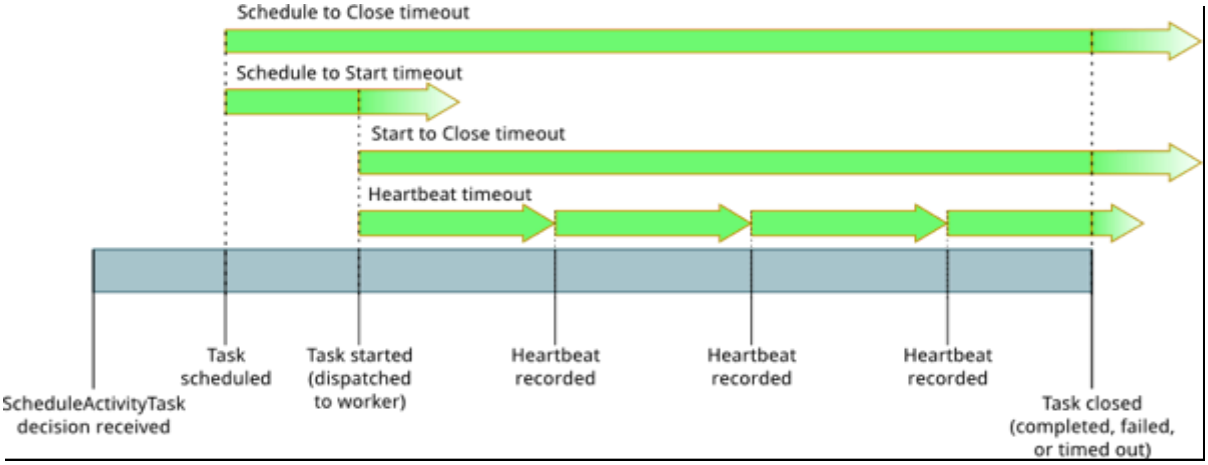
How to Design Workflow Ssytem

http://www.raychase.net/3758

Recent work has been and SWF (Amazon's Simple Work Flow) to deal with, in a SWF-based workflow framework to develop and repair the above bug. SWF's activity timeout is 5 minutes, after the activity task begins execution, activity worker needs to take the initiative to send a heartbeat request inform the service side: "I'm alive, I'm still working," if there is more than five minutes (can be configured) no heartbeat , SWF end of the service believes that you have hung up, I need to arrange this activity to another activity worker up executed. Borrowing AWSofficial website of a map:



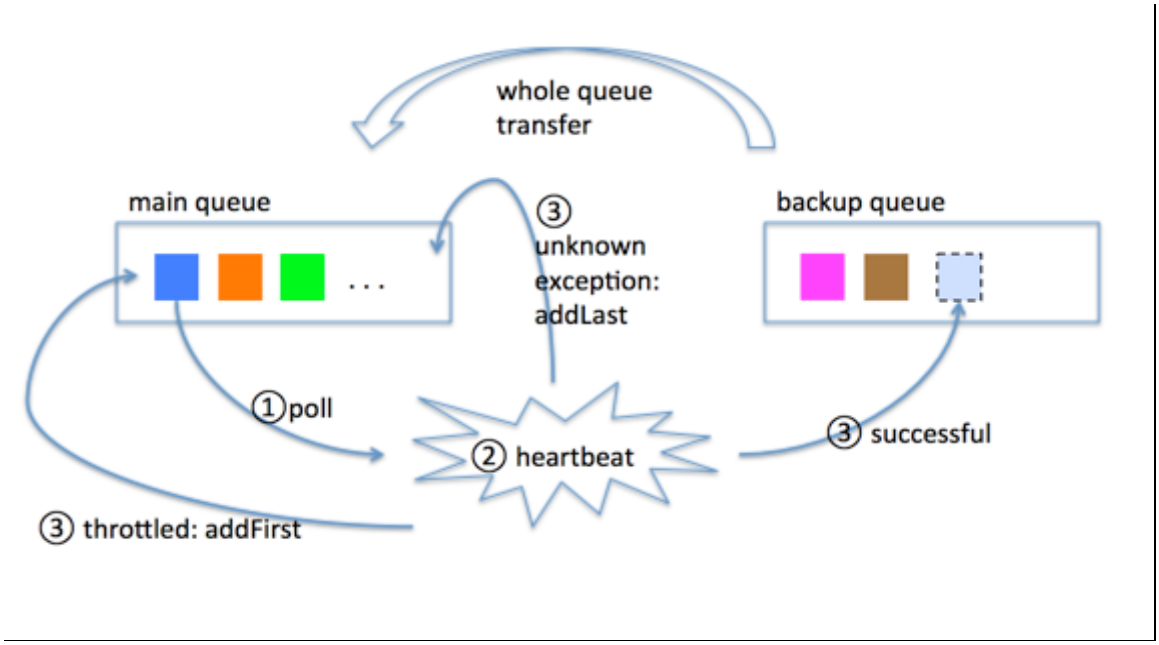
There are several activity task being executed on each machine. It can be seen, after the task activity started up with the constant need to inform the service end of the heartbeat task is still in progress, activity worker alive. This "report" requires activity host worker where the initiate, which is stateless SWF end of service (a few years ago wrote a bit describes it), one of the basic requirements. Tasks are made to pull the worker side, these behaviors are also triggered active worker side.

This mechanism is simple to describe, but the actual design and implementation of the relevant time, and there are many interesting places worth pondering.

In this workflow inside, I do heartbeat mechanism is implemented as follows:

- There are two queue, is a main queue, the dequeue (double-ended queue); the other is the backup queue, the general queue. Both are used to store information needed to send the heartbeat of activity (heartbeatable objects).
- Every second attempt to perform such a method A: From the main queue inside a poll heartbeatable objects (if the queue is empty ignore this execution), check the heartbeat represents activity task if still working, if so, to send a heartbeat. After sending successfully, put the heartbeatable objects thrown into the backup queue to go inside. Thus, a one second, progressively, main queue of all objects heartbeatable slowly being transferred to the backup queue gone.
- Every two minutes (called a cycle) to perform Method B: the backup queue inside all heartbeatable objects all transferred to the main queue to go, so he can continue to perform above step-by-heart logic.

The basic advantage of this mechanism is that all activity task heartbeat unified management usually does not guarantee a rapid heartbeat (a default configuration of one second, or does not send), while ensuring that no one will be missed:



However, there are many, many problems will emerge:

Why use two queue?

First, there is this fact: A method in the course of implementation, in theory, it will be executed once per second, but there is no compulsory guarantees, so before the second execution of A will be completed before the second A starts. In other words, their theoretical start time is sequential, but the actual start time and the actual execution time of the heartbeat is uncertain, the situation is complicated by the need to deal with. And in the end may be configured to perform A number of parallel threads, depending on the function of the heart for this thread pool exists at most. Thus, in the judgment and execution process, the need for the current poll out heartbeatable object locking.

Two queue, mainly in order to record in this cycle which can easily determine whether an object has completed a heartbeatable heartbeat behavior. Heartbeat has not been completed, are in the main queue in; we finished, are placed in the backup queue.

If you use a queue, it is also a solution:

- There is a public counter, when the beginning of each cycle, to counter +1.
- Each heartbeatable object itself needs to carry a private counter that identifies the current round of the heartbeat cycle has been completed.
- heartbeatable subject to the completion of each after its own counter +1 throw the tail; from each team to take the first take A new heartbeatable objects of the time.
- If you get to the object itself equal to the value of the counter has a public counter, tell the whole queue inside the target heart rate have been completed.

Of course, the drawbacks of this approach is that, to determine whether the heartbeat also need to send this matter, not only in taking objects from the queue, but also to determine the

object's counter value, complexity and overhead significantly larger than two queue solutions.Because the next two queue solution, just need to try to take the object from inside the main queue like, taking less than an object of this cycle there is no need to send a heartbeat to illustrate. It seems to be more of a queue, but program it is still simple.

Heartbeat frequency is kept in the long as well?

Obviously not better, not just costs, but also because heartbeat needs to consume resources, such as CPU resources; and, in the service end of the heartbeat also throttling, current activity worker initiated too frequent heartbeat, the heartbeat of the current may be refused, may also let other activity worker's normal heartbeat rejected.

The core problem we have to solve is, under normal circumstances, must be able to cap a successful launch within five minutes like a heartbeat.

To say, as much as possible to increase cycle, that every five minutes I designed a timer on the implementation of a fine. But the problem is not so simple, first of all to consider initiating the heartbeat is not necessarily successful. If only to try to initiate the heartbeat in approximately 5 minutes when, if they fail, there is no time to re-try. Therefore, to trade-off.For example, the configuration cycle is 120 seconds, this benefit is, within 5 minutes timeout, can cover 1 to 2 full cycle. If the cycle configured for three minutes, then five minutes can not strictly guarantee covering a complete cycle.

Determine heart rate are two important parameters, one is the execution frequency, Method A, a is the length of time of a cycle. For example, the former is 1 per second, which is 2 minutes, and then in the ideal case, a cycle 120 seconds, and can handle 120 activity task, in other words, the limit is 120 activity task performed together on this machine. More than this number, it means that in one cycle, unable to complete all the tasks for sending heartbeats.

Of course, the actual situation is not so satisfactory, taking into account the transient network issues, threads, CPU resources, competition, etc., can actually parallel activity task a lot lower than this number.

Exception handling and retry

In the figure, there are three steps ③ arrows indicate there are different kinds of processing heartbeat situations:

- There are some general exceptions, for example, it means that the resource does not exist, or cancel the task has been, and when this happens, take the appropriate activity task to cancel out, while the object itself this heartbeatable permanently remove the queue.
- Retry case 1: throttling caused by abnormal, when such an exception occurs, the current heartbeatable objects back again addFirst main queue, because this is not what is currently unsolvable problems or unexplained cause, that is, simply try again can.
- Retry Case 2: other unknown causes of abnormal, of course, this situation needs to be retried (before we lack such a retry mechanism, leading to the next activity task can get a chance to thrust into the heart of a cycle, which is obviously not enough reasonable), but you can put heartbeatable object into the queue to the tail of retries (addLast), and accompanied by a private counter, if more than a certain number of retries, then moved to the next cycle (backup queue) to go. The tail of the way into the queue, so you can try again in the current cycle, so that they can try again this can minimize the impact on other

objects heartbeatable heartbeat timely delivery. In fact, the entire process is to retry the current failed objects and then back into the queue process, no threads are blocked.

I have met some of these issues, and improved only the above-described mechanism:

In the CPU or the load reaches a certain degree of time (for example, at this time there is a process call service, taking up a lot of CPU resources), it is prone to heart problems can not be timely, such as sometimes thread has been initialized, but will be stuck some time, because there is not enough resources to carry out. Wait until a certain time, the resources are released (such as the end of this call service process), before the accumulated time heartbeat task will suddenly explode. Not only can not guarantee the order in which the heartbeat, and severe cases can lead to throttling. If there is no retry mechanism within the current cycle, the next heartbeat of the object need to wait until the next cycle, it is likely to cause activity task of timeout.

Besides following a heart abnormality and related issues.

In one case, within the framework of this workflow, we need to manage EMR resources, there is an activity to EMR cluster initialization is complete, another activity of the actual implementation of the steps to submit up. But found in actual operation has the following problems: EMR cluster has been initialized, but steps has yet to submit way up, leading to the cluster idle too long, monitor within the framework are considered to have no use, and need to be recovered, so it was terminate the EMR cluster. But after that, steps up was only submitted, but this time the cluster is already in terminating state, and naturally this step Submit failed.And after analysis, the EMR cluster causing unintended termination, including several reasons:

- decision task timeout. After EMR cluster is created, SWF will ask why the next step decider, this time for various reasons, if high CPU load, leading to decision task timeout, SWF, etc. would have been in there, and if this time timeout feature too too long, this time-out was enough above the EMR cluster idle for too long leads to mistaken recovered.
- Analyzing EMR cluster to a certain idle time will recycle logical problems. Our previous implementation is executed once every 2 minutes "EMR Resource Operations," including checking resource status, resource operations, and then create a resource if the EMR found after four resource operation, still did not step up to submit, they think idle too long and need to be recovered (2 x 4 = 8 minutes). But the problem is actually due to various reasons (execution interval and the heartbeat of the actual time of uncertainty same principle), intervals EMR resource operation does not strictly Guarantee 2 minutes, sometimes every period of time are not enforced, and sometimes It will usher in a concentrated outbreak, this time it might actually EMR resources idle for far less than 8 minutes to be recovered. Therefore, this logic it is better to use absolute "free time" to determine, for example, recording a good time EMR resource creation, all with the current time and after every test to compare creation time, idle more than eight minutes and then recovered.
- Since we mentioned before can not be completed in time lead to heart activity task timeout, then the EMR cluster actually created the task has been completed, but is treated as overtime to ignore.

Finally, I want to say yes. A good design workflow framework, there are still many difficult places, especially the need thoughtful place. Even so SWF-based workflow to an existing building blocks and overlay functionality, there are a lot of difficult and interesting places.

http://www.raychase.net/3998

Scalability

Basically just what infrastructure design, scalability, are important considerations. As workflow is concerned, substantially horizontal expansion work node is considered the most important indicator of scalability. Since work node can scale horizontally, then this means that the task (task) must be based on the initiative to pull way acquired by the worker nodes, rather than pull way to allocate from the scheduling node (once very briefly compared the pull and pushbut in fact, the difference is far more than the contents of plain text). The distribution of tasks, need to consider something like this: If there is more work to try to pull node task that is assigned to whom? Specifically, for example, such an example: If each task node allows to perform five tasks simultaneously, while the total number of tasks can be executed simultaneously now only five, there are a total of five task node, the ideal state should be five one is evenly distributed to five nodes go, but with a simple pull mechanism can not guarantee this, it is possible that all five tasks went up a machine, because it does not exceed the number of nodes can simultaneously perform a task It limits.

On the other hand, generally speaking, all tasks should be idempotent, that can be repeated to the Executive, performed several times and performed once the result is the same.Implementation tasks node error can occur at any step, as the number of nodes increases, such errors become more of the norm rather than "abnormal." Health worker nodes need to maintain and in some way by the notice, the most typical and cheap and effective way is the "heartbeat"

Functional Decoupling

- Decoupling resource management and task management. Which I only saw a few workflow inside. Task management workflow are almost all available, but a separate resource management is not. For example, I could write a task to perform EMR tasks, you can also write a task to perform on the EMR, EMR implementation management logic, the code can be shared by our way - but this architecture, you the task is difficult and my task safely and efficiently share the same EMR resources, whether the resources to create, destroy, query status, or throttling, have become very troublesome. Similar examples are shared database, printer sharing, or even additional share a workflow system. When there is a large resource overhead, we often need to be unified workflow management level up, or manage a few resources, but shared to a large number of task.
- Decoupling business logic and scheduling logic. This basically includes all workflow inside, scheduling logic is independent of the business is relatively "dead" thing, state management, workflow, and the success of each task failure. But the business logic and workflow is composed of one of the task "living" in flesh and blood. I have not seen what the business code and workflow scheduling logic written together.
- Decoupled status inquiry and scheduling system. A complete workflow system, scheduling is only one aspect in which the core, if not a good state inquiry system, the maintenance workload will be enormous. And both must be decoupled open. For example,

the state of the workflow and task execution, must be in some kind of persistent storage medium, such as relational databases, such as NoSQL databases, such as disk log files, etc.This time, the scheduling system can be said that the information written to the storage system, the most important source, which reads the information, it is possible to read from the scheduling system, may also be read from the status inquiry system. Format or schema of the stores must be relatively stable. This consistency and availability of storage, the entire system will be a core component of consistency and availability.

- Decision-making and execution systems decoupling. Decision-making system for determining whether a task to meet the conditions and begins the execution, which is the brain of the whole workflow system; the implementation of the system is a specific task, it is the flesh and blood throughout the workflow system.
- Decoupling event system and monitoring system. Involved in this workflow minority.Many workflow systems have internal event system, such a task is assigned to a node, and a task execution fails, etc., but such an event monitoring system, but not independent, leading to follow-up for a special event to be executed specific logic becomes difficult.

Synchronous and asynchronous tasks

In fact, when taking into account the independent resource management division of asynchronous and synchronous task becomes natural.

- There are many tasks to be performed on the current work node. For example, you need to download a file in the working node, and then go through later processing written to the database, these tasks consume large amounts of memory and CPU, the need to allocate a separate dedicated thread to complete, is the synchronization task.
- There are a number of tasks, work node is not the actual work performer, but one resource for client systems only responsible for submitting to the task within the system and is responsible for management and monitoring. For example, the print job, submit a print request to the printer, and then only need to constantly check the status of the task to the printer, and to make and delete tasks and other operations required to re-submit. These tasks usually do not require long-term possession thread, a thread can handle multiple tasks in a single cycle. They are asynchronous tasks. In addition, for a special case, nested workflows, ie workflows call child workflow, then the sub-query workflow status of this action, it must be asynchronous tasks. Asynchronous tasks involving notification and monitoring mechanism of the event, later mentioned.

Distributed Lock

In some cases, distributed lock becomes a necessary option. Such as the aforementioned resource management. There are many resources are requested operation is exclusive, in other words, two operations are not supported concurrent calls, not unforeseen problems may arise during; on the other hand, a node when the resource to operate, it needs to collaborate with other nodes so as to operate two working nodes is ordered and right, and will not conflict.

For example, work node A to query the current state of EMR, if has been idle for 10 minutes, will perform the operation to end off the EMR resources; working node B then query the status of the EMR, if not to be finalized, will to submit a new computing tasks to the above. At this time, if there is no collaboration distributed lock, the question came up, the Node B may be a

state inquiry found that EMR is still alive, that this moment on, A node end of it, but B does not know, then submitted a computing tasks to this we are over the (dead) EMR resources, so this is bound to perform computing tasks submitted failed.

There are many implementations of distributed lock, strong consistency simple storage system, of course, have a more efficient implementation, such as some specialized distributed lock system.

Function scalability

Before talking about the scalability, performance architectural, functional level and vice versa.

- Custom tasks. This is what almost all workflow systems will be considered, which is the decoupling of business logic and scheduling logic of necessity. Because when the workflow system design, can not necessarily predict all the type of task, users can define their own implementation of logic.
- Custom resources. With resource management, it is necessary to define the resource from.
- Custom event listener. Event management is usually in the workflow system is very easy to overlook the content, for example, I want to send a special message informed me, which need to monitor this event offers the possibility of expansion in one task time-out.
- Workflow task runtime execution conditions. Typically workflow defines how a document will have to perform the (meta file), but there are a number of parameters and conditions of execution can be determined at runtime, and even depend on the results of the implementation of the previous step, or the need to perform some logic to get.

Availability and reliability

Most workflow, have adopted the design to the central node, to ensure that there is no single point of failure. Are all subsystems. Also ensure that in the case of business pressures increase, marking the availability of the latency within the expected range. Other content is not expanded, this presentation of articles everywhere.

Lifecycle Management

Here refers both to a lifecycle management workflow execution, also refers to the life-cycle management of a single task.

Talk about these necessarily involves the following questions:

- workflow definition and workflow separation, task definition and task execution separation. Wherein the definition define the execution logic, and execution environment and execution really, time, etc. related parameters. Logic can usually only one (but not necessarily, depending on workflow support multiple versions, later mentioned), but with the execution retry occurs, save multiple copies.
- When the workflow retry process parameter variations. Change some parameters will not affect the mission has been completed, but some parameters are not.
- When the workflow retry processing for completed tasks. We hope that some cases have been re-execute the job done, but the situation we hope to complete the task they have been skipped.

- Retries task, and back off when policy retries. For example, first retry to wait five minutes, the second retry to wait 10 minutes and try again up to 2 times.
- How to politely end the task execution on the working node. In many cases we had to interrupt and end the task execution on a node, such as node need to restart the work, this does not count as a result of the implementation of the code of business mission failure, but rather a "resource termination". In this case, the task often need to be assigned to a node to another living, but here there is the reallocation of the strategy involved, as already mentioned.
- Right heavy task. Or called priority, this feature I saw only a few of the workflow. Taking into account the allocation of resources, some of the more important tasks with a higher priority, and even trivial task failure does not affect the status of the workflow.

Task DAG design and expression

This is a flowchart of workflow execution, it is also dependent on the expression of the relationship between all the task. I have seen a variety of means of expression, there is XML, but also, there are some unknown own definition of the JSON format. Some of workflow definitions can be a graphical tool to help accomplish this flowchart. The DSL design, to some extent determines the workflow is not able to use easy to understand. Also mention that this optional graphical tools mentioned here, after all, is only a secondary, it is not the core workflow (DSL you can say that this is part of the core, but this is obviously not a tool to help complete) - My Opinion had a team, workflow overall design Debu how, run up a bunch of questions, but this tool spent a lot of time and effort to repair, upside down.

In addition, the workflow status and implementation, as well as their archiving and management, but also need to integrate a tool to assist. Almost all aspects of this workflow are available, usually web tools, and command line tools.

Manage input and output

It is also a nice-to-have stuff for every task, there are input and output, they can be completely handed over to the users themselves to achieve, such as the user store them to a file inside, inside or written to the database, and workflow fundamental regardless, each task inside yourself to read the corresponding user file. However, a better approach is, for some common and simple input, output, is together with the execution of the workflow and task persistence of state to go inside. The definition would also facilitate the workflow inside, put some expression based on the results of the previous step task to perform follow-up decisions.

In addition, there is a slightly less popular use case, is the input and output management.Typically workflow is repeated, and the data size of each execution of the input and output is often a lot of people care about. In this section, I have not seen any workflow provide such functionality. Many users write their own scripts and tools to obtain such information.

Independent metrics and logging system

For metrics, the core content is nothing more than the health of the nodes, CPU, memory, task execution time distribution, the failure rate and so few. In some cases users also want their own extensions.

About logs, mainly it refers to the archiving and consolidation. Archive, referring to the history log is not lost, not lost or within a certain time, expired logs can be overwritten, so as not to cause the disk capacity issues; and combined, referring to the log can be a more unified perspective to query and browse , a problem will not go to each machine to manually locate.The lack of this feature, sometimes a lot of trouble. At work I encountered a problem of resources is aborted, in order to find the node ending resource, I consult dozens of log nodes, painful.

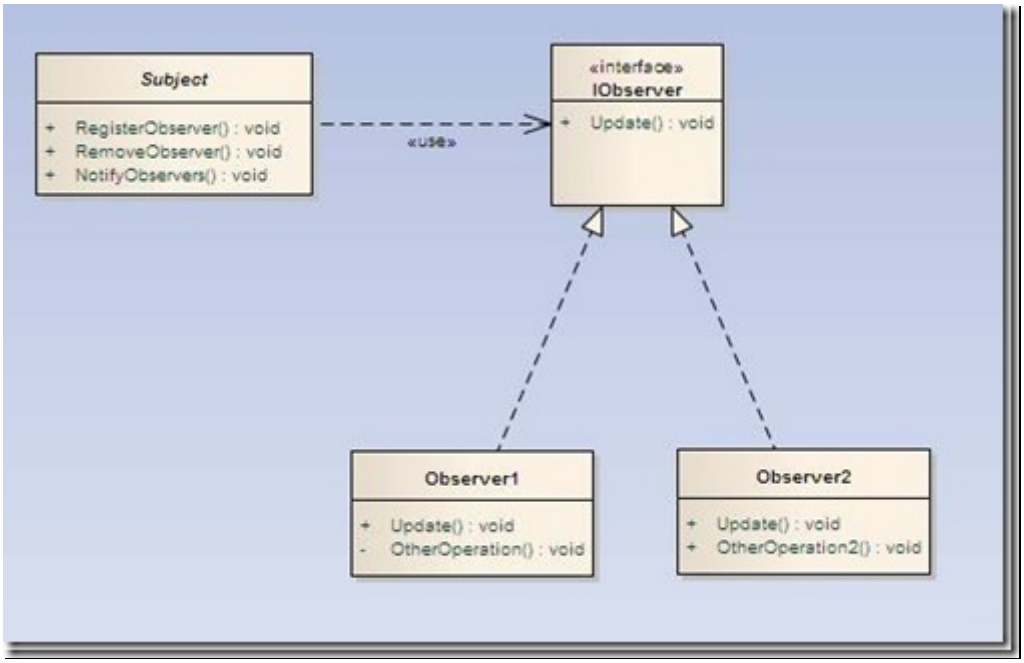
Version control and smooth deployment

Put these two is because the code upgrade is inevitable and often occur together. In order to ensure smooth deployment, apparently Normally, the code can not be updated simultaneously on the node, part of the required part. For example, the first 50% of the termination node, after deploying the code to activate and ensure successful, then the remaining 50% of that node.However, there exist problems of old and new codes in this period, which usually brings a lot of bizarre problems. For this problem, I've seen two solutions:

- One is the simultaneous deployment of all nodes, all the nodes in this case all of inactivation, there may be caused because the inactivation task timeouts, and even lead to workflow execution failed. But workflow lifecycle management by a separate dispatch system, therefore removing the outer most part unaffected timeout.
- There is also a part of the part of the deployment, the smoothest, but in this case the need to manage multiple versions of coexistence, but also the quality of the code proposed new requirements - backward compatibility.

No matter what kind of choice, this approach is relatively simple to implement, but there are many problems, such as in this case, how to deal with external resources? For example, performed on the external EMR resources Spark task, but there are old code has been implemented into the EMR up, this time updating nodes, these tasks are performed EMR how to deal with? It is invalid or retained, if you leave, then they can still rely on the implementation of the old code, whether the result of the subsequent processing and the new code will be immediately deployed in conflict. Another example for change (such as a change DAG) workflow defined on the existing execution, should be how to deal with, update or intact (usually remain the same, because the update complex problems caused by very much).

http://www.raychase.net/244
observer mode, the messages are relatively push and pull way to pass the
Update yourself.



Now say the differences here:

"Push" mode means, Subject maintain a list of observers whenever an update occurs, Subject will take the initiative to push the update message to each Observer.

"Pull" approach means that individual Observer maintains Subject list their concerns, to decide at the appropriate time to obtain the corresponding Subject of update data.

"Push" benefits include:

1, efficient. If no updates occur, there will be no updates to push the action, that is, each message push occurred after indeed update events are meaningful.

2, real-time. The first time after the event to trigger the notification operation.

3, may be established by the Subject of the notification time, you can avoid some busy time.

4, can express a different sequence of events.

Good "pull" include:

1, if the number of observers, Subject to maintain a list of subscribers, can be difficult or cumbersome, the relationship between free subscribe to Observer to complete.

2, change events Observer can ignore it does not care, just to go get their own interest to the event.

3, Observer can decide for updated event time.

4. Subject to pull form allows better control of each Observer update each query access.

In fact the "push" and "pull" can be compared too much content, such as:

The client is usually unstable, the server is stable, if the message is initiated by the client to obtain, it is very easy to find the address of the service side, it is easier to do access control (focus on the service side one), the service end may be relatively easy to track the location and status of the client, not vice versa;

Access to the Internet page is the best example of a "pull" mode;

We want to stress is usually distributed to each client up, the server only the core of the matter, to provide content only, does not manage distribution lists;

.....

Another idea is a form of "push" and "pull" a combination of, for example, the server is only responsible for notification of a number of data is ready, as and when the need to get the client to obtain these data, entirely by the client themselves determine.