Асинхронное взаимодействие

Java Execution Framework
Сравнение производительности
Асинхронные вызовы в Glassfish

Кислин Григорий

АСИНХРОННЫЕ ВЫЗОВЫ

Execution framework (java.util.concurrent)

Yota

A task that returns a result and may throw an exception

```
public interface Callable<V> {
    V call() throws Exception;
}

public  interface Runnable {
```

Future represents the result of an asynchronous computation

```
public interface Future<V> {
    boolean cancel(boolean mayInterruptIfRunning);
    boolean isCancelled();
    boolean isDone();
    V get() throws InterruptedException, ExecutionException;
    V get(long timeout, TimeUnit unit)   throws InterruptedException,
ExecutionException
}

public interface ScheduledFuture<V> extends Delayed, Future<V> {
}
```

Execution framework (java.util.concurrent)

Yota

ScheduledExecutorService： Executors.newScheduledThreadPool(int corePoolSize)
newSingleThreadScheduledExecutor()
ExecutorService: Executors.newCachedThreadPool()
Executors.newSingleThreadExecutor
Executors.newFixedThreadPool(int nThreads)

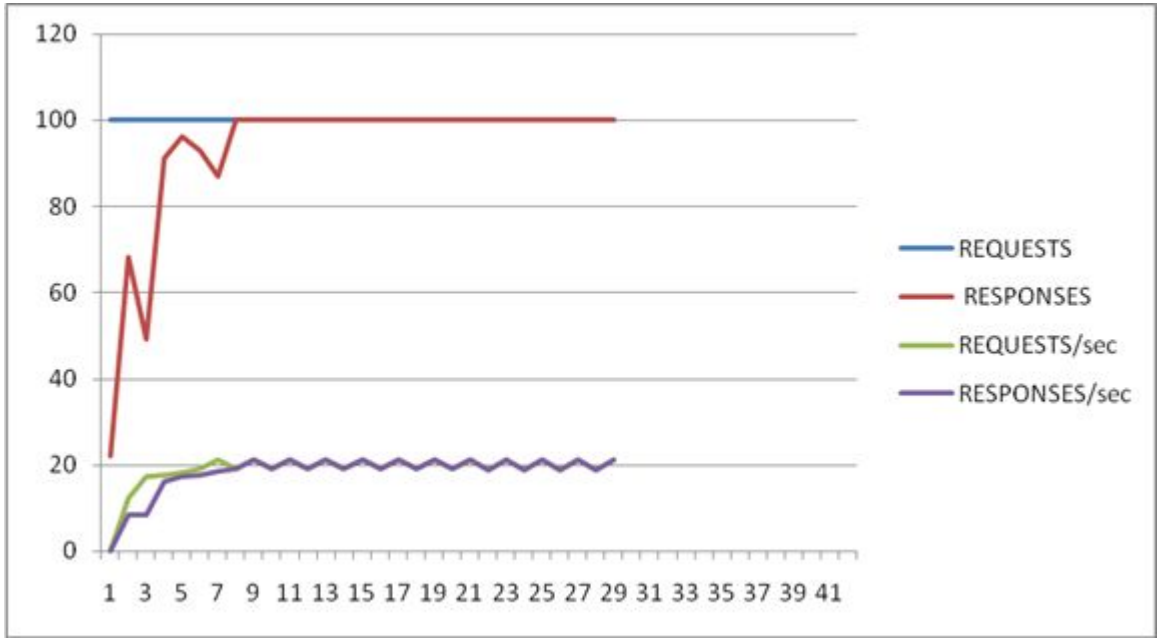
BlockingQueue
Implementation: LinkedBlockingQueue

```
public interface CompletionService<V> {
    Future<V> submit(Callable<V> task);
    Future<V> take() throws InterruptedException;
    Future<V> poll();
    ...
}
```

Implementation: ExecutorCompletionService(Executor executor)

Сравнение производительности

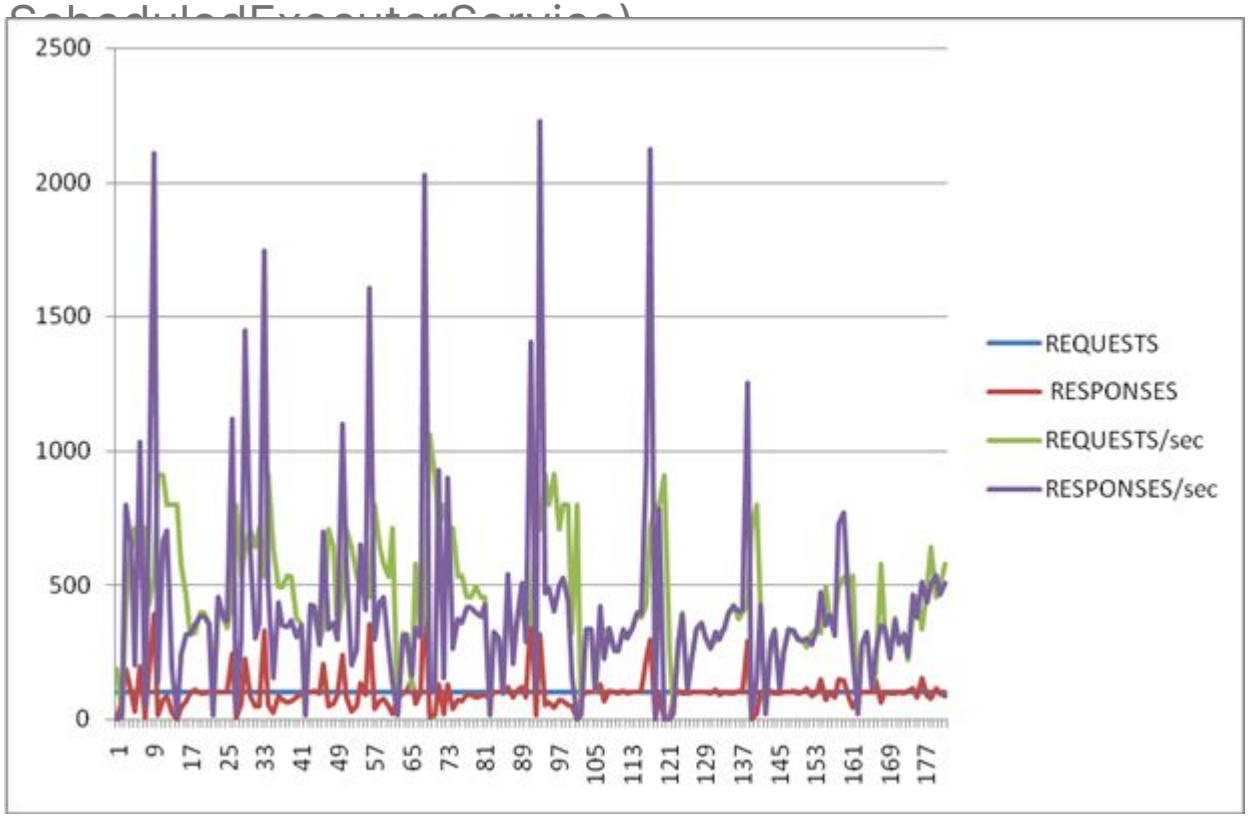
Синхронный сервлет (веб-сервис), ожидание 10 сек



Сервлет исполняется 10 сек и удерживает все это время свой поток а в пуле 200 потоков, то нагрузка 200/10 сек = 20 запросов/ сек.

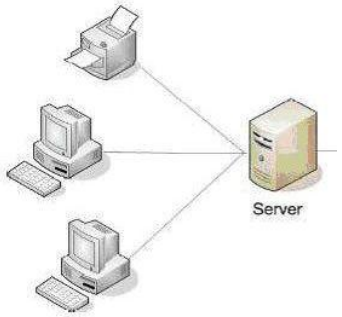
Сравнение производительности

Асинхронный сервлет, ожидание10 сек (GlassFish v3, ScheduledExecutorService)



Нагрузка- примерно 500 запросов/сек. (возможно ограничение на стороне 1 клиента)

При длительности в 10 с и 200 потоках в пуле – выигрыш в 25 раз



АСИНХРОННЫЕЙ КЛИЕНТ В GLASSFISH

Glassfish
асинхронный клиент веб-сервиса

Yota

wsimport

```
wsimport  -b async.xml

        <bindings node="wsdl:definitions">
                <enableAsyncMapping>true</enableAsyncMapping>
       </bindings>

@WebMethod
@RequestWrapper(localName=..
@ResponseWrapper(localName =“Result”, ..

 public Response<Result> execute();

 public Future<?> execute(
          @WebParam(name = "asyncHandler", targetNamespace = "")
          AsyncHandler<Result> asyncHandler);
```

Glassfish
асинхронный клиент веб-сервиса

Yota

Пример

```
public interface Response<T> extends Future<T> {
      Map<String,Object> getContext();
}

public interface AsyncHandler<T> {
       void handleResponse(Response<T> res);
}

 final AsyncHandler<Result> handler = new AsyncHandler<Result>() {
          @Override
          public void handleResponse(Response<Result> res) {
                    res.get() ....
          }
```

Glassfish
асинхронный клиент веб-сервиса

Yota

+ есть в Metro 2.1 (Glassfish 2)
+ достаточно прост и удобен в реализации
+ генерируется из wsdl, возможно выборочное применение к разным методам.

Glassfish
асинхронный клиент веб-сервиса **@Oneway**

Yota

Пример

```
<operation name=« execute">
    <input message="tns:Result"/>
</operation>
```

```
@Oneway
@WebMethod
public void execute()   no exceptions
```

Glassfish
асинхронный клиент веб-сервиса **@Oneway**

Yota

+ есть в Metro 2.1 (Glassfish 2)

+@Oneway очень прост в использовании (просто возвращается акноледж- статус код 202, который обрабатывается jax-ws)

- при необходимости получения результата или Exception необходима реализация callback и его обработки на стороне клиента (передавать url для callback можно в через WS-addressing), либо result polling.

Glassfish v3
асинхронный клиент EJB

Yota

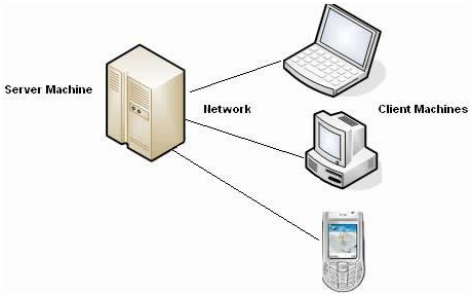
```
@Remote
public interface IAsyncEJB {
    @Asynchronous
    public Future<Result> execute() throws Exception;
}

public class AsyncEJB implements IAsyncEJB {
    @Asynchronous
    public Future<Result> execute() throws Exception{
        ...
        return new AsyncResult<Result>(res);
    }
}
```

Glassfish v3
асинхронный клиент EJB

Yota

- появились только в EJB 3.1 (Glassfish v3)
-недоступны как веб-сервисы
-неудобны в использовании (возвращается удаленная реализация Future)
-при нагрузке работали очень задумчиво



АСИНХРОННАЯ РЕАЛИЗАЦИЯ СЕРВИСА В GLASSFISH

Glassfish
низкоуровневая реализация веб-сервиса

Yota

```
@WebService
public class AsyncWebService implements AsyncProvider<Source> {


   public void invoke(final Source request, final AsyncProviderCallback<Source> callback,
    final WebServiceContext ctx) {
      Executor.submit(new Callable<Void>() {
                  @Override
            public Void call() throws Exception {
               try{
                              …. // parse request, form result
                  callback.send(result);
                        return null;
                   } catch (Exception e){
                  callback.sendError(e);
```

Glassfish
низкоуровневая реализация веб-сервиса

Yota

- + появились в Metro 2.1 (Glassfish v2)
- + большая гибкость в использовании
- - длительность разработки (необходимо реализовать единственный метод invoke - самостоятельно маршалить-унмаршалить запрос-ответ, определять имя операции, делать web.xml и sun-jaxws.xml, отсутствует в GF консоли)

Glassfish v3
асинхронные сервлеты

Yota

```
@WebServlet(urlPatterns = {"/async"}, asyncSupported = true)
public class AsyncServlet extends HttpServlet {
    private static final ScheduledExecutorService executor =
   Executors.newScheduledThreadPool(1);

      @Override
       protected void doGet(HttpServletRequest req, HttpServletResponse res) {
       final AsyncContext ac = req.startAsync();
       Executor.submit(new Callable<Void>() {
              @Override
              public Void call() throws Exception {
                  HttpServletResponse res = (HttpServletResponse) ac.getResponse();
            ….
            ac.complete();
            return null;
              }
```

Glassfish v3
асинхронные сервлеты

Yota

+ легко писать, удобно использовать

- появились только в Servlets 3.0 (Glassfish v3)

- в основном взаимодействие происходит не через сервлеты