



Асинхронное взаимодействие

Java Execution Framework

Сравнение производительности
Асинхронные вызовы в Glassfish

Кислин Григорий



АСИНХРОННЫЕ ВЫЗОВЫ

Execution framework (java.util.concurrent)

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)

ScheduledExecutorService : Executors.newScheduledThreadPool(int
corePoolSize)

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

BlockingQueue

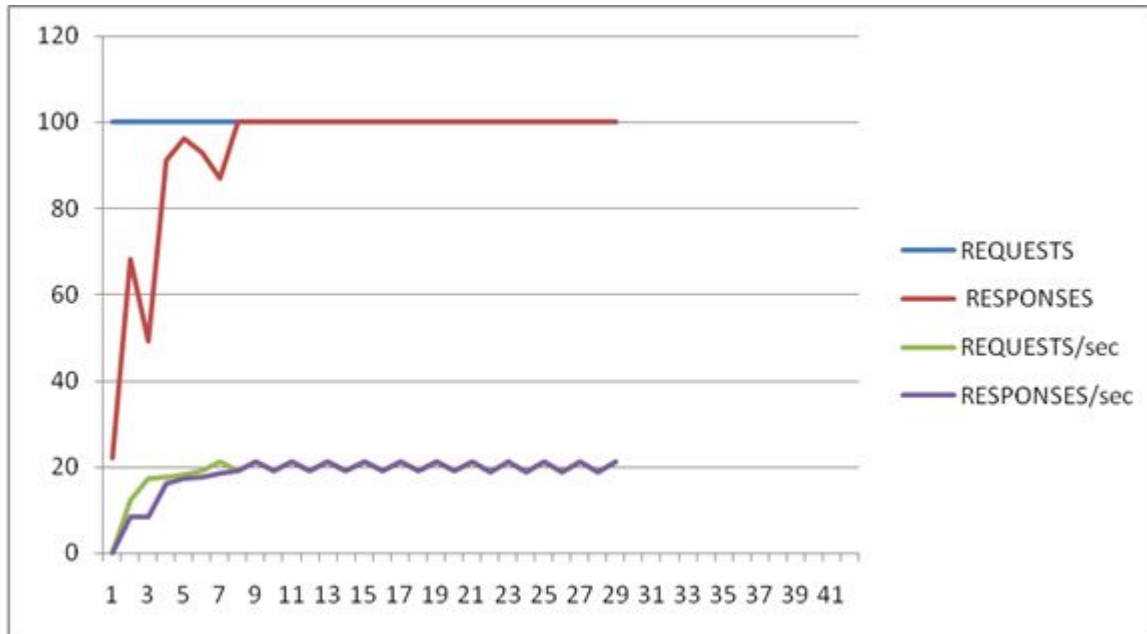
Implementation: LinkedBlockingQueue

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

Implementation: ExecutorCompletionService(Executor executor)

Сравнение производительности

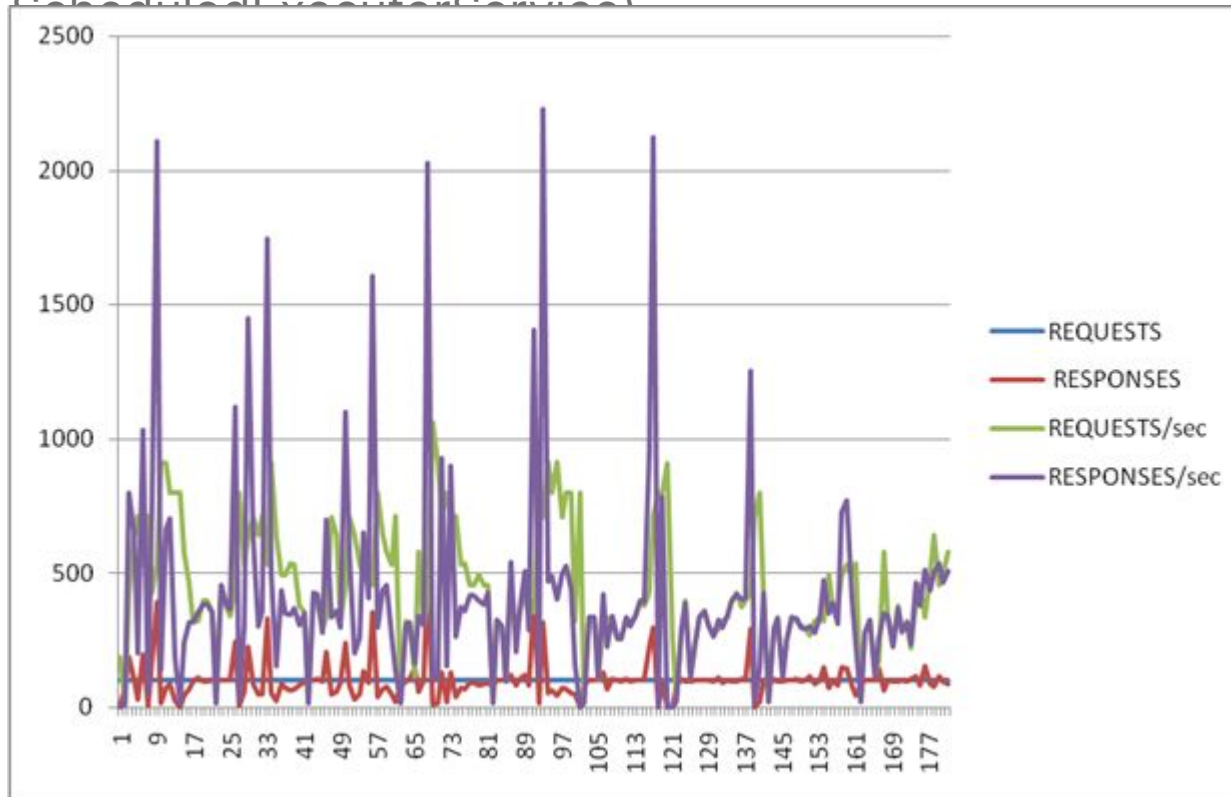
Синхронный сервлет (веб-сервис), ожидание 10 сек



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

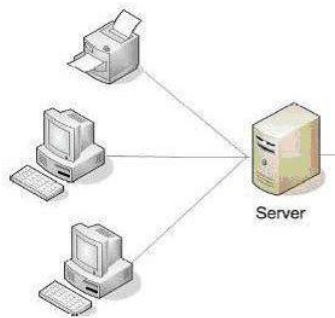
Сравнение производительности

Асинхронный сервлет, ожидание 10 сек (GlassFish v3, ScheduledExecutorService)



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

При длительности в 10 сек и 200 потоках в пуле - выжили в 25 проц



АСИНХРОННЫЙ КЛИЕНТ В GLASSFISH

Glassfish

асинхронный клиент веб-сервиса



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);
```


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() ....  
    }  
}
```

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

Пример

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

@Oneway

@WebMethod

public void execute() no exceptions

- + 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

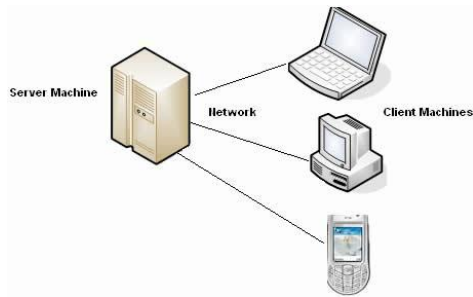
асинхронный клиент EJB

@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);  
    }  
}
```

- 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

```
@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);
```


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

Glassfish v3

асинхронные сервлеты



```
@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



- + легко писать, удобно использовать
- **появились только в Servlets 3.0 (Glassfish v3)**
- **в основном взаимодействие происходит не через сервлеты**