
Google™
11
I/O

Highly Productive GWT

Jeff Schnitzer, David Chandler, Philippe Beaudoin
May 10-11, 2011

Hashtags: #io2011 #DevTools
Feedback: http://goo.gl/Rj0NA

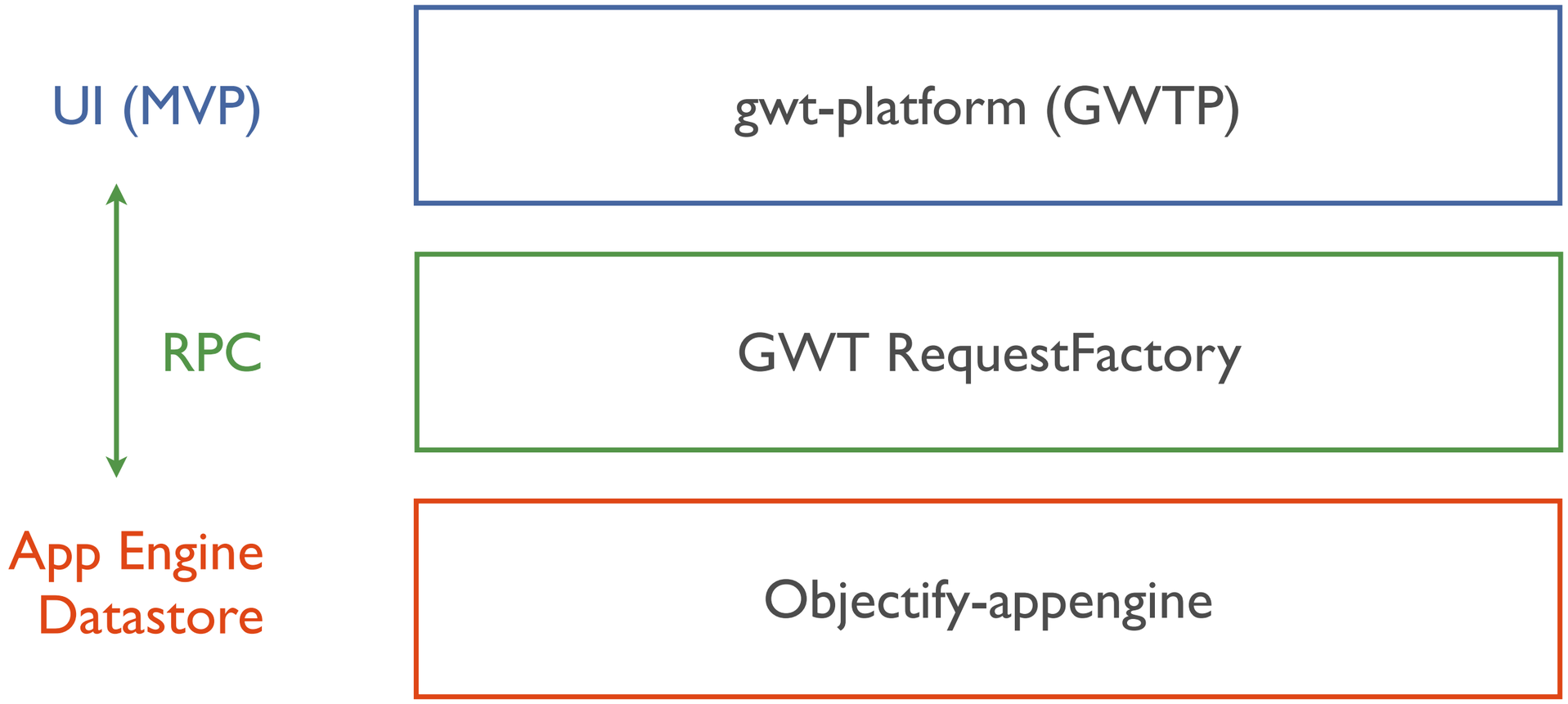
Agenda

- Some useful 3rd party tools for GAE+GWT apps
- Intro to Objectify (Jeff Schnitzer)
- Using RequestFactory with Objectify (David Chandler)
- Intro to gwt-platform (Philippe Beaudoin)

Some 3rd party tools for GAE and GWT apps

- Maven!
  - File | Import existing Maven project w/ Google Plugin for Eclipse (and m2eclipse, m2extras)
  - maven-gae-plugin
  - gwt-maven-plugin
- App Engine Datastore persistence
  - Twig
  - Slim3
  - Objectify
- GWT MVP (Model View Presenter) architecture
  - mvp4g
  - gwt-presenter / gwt-dispatch
  - gwt-platform

A highly productive app stack



gwt-platform

RequestFactory

Objectify

Introduction to Objectify-Appengine
Jeff Schnitzer

Objectify-Appengine

The simplest *convenient* interface to the GAE Datastore

http://objectify-appengine.googlecode.com/

What's wrong with JPA/JDO?

- Extra cruft
  - Fetch groups
  - Detaching
  - Owned/unowned relationships
  - Sophisticated query languages

- Leaky abstraction
  - Keys
  - Entity groups
  - Indexes

Too much to learn!

- How should data be modeled in JPA/JDO?
- How do you configure DataNucleus?
- What parts of DataNucleus work on Appengine?
- How should data be modeled in native datastore?

Too much to learn!

- How should data be modeled in JPA/JDO?
  - Read the spec docs
- How do you configure DataNucleus?
  - Read www.datanucelus.com
- How what parts of DataNucleus work on App Engine?
  - Read Appengine documentation - maybe
- How should data be modeled in native datastore?
  - Watch Google I/O videos

Too much to learn!

- How should data be modeled in JPA/JDO?
  - Read the spec docs
- How do you configure DataNucleus?
  - Read www.datanucelus.com
- How what parts of DataNucleus work on App Engine?
  - Read Appengine documentation - maybe
- How should data be modeled in native datastore?
  - Watch Google I/O videos

Datastore Basic Operations

- **get**
- **put**
- **delete**
- **query**

Working with the Low Level API

```
Entity ent = new Entity("Car");
ent.setProperty("color", "red");
ent.setProperty("doors", 2);
service.put(ent);
```

Give us objects!

```
Car car = new Car();
car.setColor(Color.RED);
car.setDoors(2);
service.put(car);
```

An Objectify Entity

```
public class Car {
    @Id Long id;
    Color color;
    int doors;
}
```

get()

```
Objectify ofy = ObjectifyService.begin();

Car car = ofy.get(Car.class, 959);

Map<Long, Car> cars =
    ofy.get(Car.class, 959, 944, 924, 911);
```


Google I/O 11

put()

```
Objectify ofy = ObjectifyService.begin();

Car car = new Car(Color.RED, 2);

ofy.put(car);

System.out.println("Generated id is " + car.getId());

List<Car> cars = makeOneThousandCars();
ofy.put(cars);
```

delete()

```
Objectify ofy = ObjectifyService.begin();

ofy.delete(Car.class, 959);

Car car = // ...get car from somewhere
ofy.delete(car);
```

query()

```
Objectify ofy = ObjectifyService.begin();

Query<Car> redCars =
    ofy.query(Car.class).filter("color =", Color.RED);

for (Car car: redCars)
    giveSpeedingTicket(car);
```

Advanced Features

- Asynchronous operations
- Polymorphic entities and true polymorphic queries
- Partial indexes
- Parallel transactions
- Embedded objects, collections
- Serialized object graphs
- Automatic memcaching
- On-the-fly schema migration

gwt-platform

RequestFactory

Objectify

Using RequestFactory with Objectify

David Chandler

Introduction to RequestFactory

- Four ways of doing RPC in GWT
  - RequestBuilder with XMLParser
  - RequestBuilder with JSONParser
  - GWT-RPC
  - RequestFactory (new in 2.1)
- Why another RPC framework?
  - No more serializer / deserializer per class
  - No accidental type explosions (List ==> AbstractList, AbstractSequentialList, ArrayList...)
  - GWT-RPC not persistence-aware, no built-in Command pattern

Introduction to RequestFactory

- Neither uses nor replaces GWT-RPC
- Designed for data-oriented services (though not exclusively)
- Acknowledges the DTO problem and minimizes pain with proxy interfaces
- Higher level of abstraction than GWT-RPC
  - Command pattern (Request object) provides batching, future caching
  - Enables RPC persistence layer with minimal boilerplate
  - Tracks changes to entities on the client and sends **only diffs**

RequestFactory goes mobile!

GWT client

Native Android client

com.google.web.bindery.
RequestFactory

one server impl

Getting Started with RequestFactory

- In gwt.xml:
  - <inherits name='com.google.web.bindery.requestfactory.RequestFactory' />
- In web.xml:
  - map com.google.web.bindery.requestfactory.server.RequestFactoryServlet
- DTO interfaces extend EntityProxy / ValueProxy
- Service stubs extend RequestContext
- MyRequestFactory extends RequestFactory
- GWT.create(MyRequestFactory.class);

RequestFactory Concepts

- *Entity*
  - has ID and version (not necessarily in entity class itself)
  - likely represented by a row in a table (a particular car in a fleet)
- *Value type*
  - no ID / version
  - likely represented by a column in a table (the color of the car)
- *Proxy type*
  - Client-side interface that represents an entity or value type
  - Shared between client and server
  - RequestFactory creates / tracks instances using AutoBeans
- *Service stub*
  - Service interface, extends RequestContext

Package layout

server	shared	client
Entities Entity locator(s) Service locator(s) Service impls	Proxy interfaces Service interfaces RequestFactory interface	Calls to RF services

EntityProxy

com.myapp.server.NamedList

```
@Entity
public class NamedList
{
   @Id private Long id;
   private Integer version = 0;
   private String name;
   private Key<AppUser> owner;
   private ListType listType;
   @Embedded private List<ListItem> items;
   ...
}
```

com.myapp.shared.NamedListProxy

```
@ProxyFor(value = NamedList.class,
   locator = ObjectifyLocator.class)
public interface NamedListProxy extends
EntityProxy
{
   public enum ListType {NOTES, TODO}
   String getName();
   void setName(String name);
   List<ListItemProxy> getItems();
   ListType getListType();
   AppUserProxy getOwner();
   void setListType(ListType type);
   void setItems(List<ListItemProxy> asList);
}
```

ValueProxy

com.myapp.server.ListItem

```
public class ListItem
{
   private String itemText;
   private Date dateCreated;
   // Getters and setters
   ...
}
```

com.myapp.shared.ListItemProxy

```
@ProxyFor(ListItem.class)
// or @ProxyForName("com.myapp.server.ListItem")
public interface ListItemProxy extends ValueProxy
{
   String getItemText();
   void setItemText(String itemText);
   Date getDateCreated();
}
```

Service stubs + RequestFactory interface

com.myapp.shared.ListwidgetRequestFactory

```
public interface ListwidgetRequestFactory extends RequestFactory {
   // Service stub accessor
   public NamedListService namedListService();
}
```

com.myapp.shared.NamedListService

```
@Service(value = NamedListDao.class, locator = DaoServiceLocator.class)
public interface NamedListService extends RequestContext {
  Request<List<NamedListProxy>> listAll();
  Request<Void> save(NamedListProxy list);
  Request<NamedListProxy> saveAndReturn(NamedListProxy newList);
  Request<Void> removeList(NamedListProxy list);
}
```

Using RequestFactory

- myRF = GWT.create(MyRequestFactory.class)
- Create an entity
  - taskService = myRF.taskService()
  - taskService.create(TaskProxy.class);
- Call setters (or use an Editor!)
- Invoke a method
  - taskService.persistTask(task);
- Fire the request
  - taskRequest.fire(new Receiver<Void>() {...});
- EntityProxyChange events get fired when entities change on server
  - PERSIST, UPDATE, DELETE
  - see Javadoc for ...requestfactory.shared.WriteOperation

Create and persist

```
public void persistList(String listName)
{
   NamedListService reqCtx = myRF.namedListService();
   NamedListProxy newList = reqCtx.create(NamedListProxy.class);
   newList.setName(listName);
   newList.setItems(new ArrayList<ListItemProxy>());
   newList.setListType(ListType.TODO);
   reqCtx.saveAndReturn(newList).fire(new Receiver<NamedListProxy>()
   {
      @Override
      public void onSuccess(NamedListProxy savedList)
      {
         // Refresh table
         listDataProvider.getData();
      }
   });
}
```

Update

```
public void update(int index, NamedListProxy obj, final String newName)
{
   NamedListService reqCtx = myRF.namedListService();
   NamedListProxy editable = reqCtx.edit(obj);
   editable.setName(newName);
   reqCtx.save(editable).fire(new Receiver<Void>()
   {
      @Override
      public void onSuccess(Void response)
      {
        // RequestFactory fires an EntityProxyChange event
        ...
      }
   });
}
```

Query (with object graph)

```
private void getData()
{
   // To retrieve relations and value types, use .with()
   Request<List<NamedListProxy>> findAllReq = rf.namedListService()
         .listAll().with("owner");
   // Receiver specifies return type
   findAllReq.fire(new Receiver<List<NamedListProxy>>()
   {
      @Override
      public void onSuccess(List<NamedListProxy> response)
      {
         updateRowCount(response.size(), true);
         updateRowData(0, response);
      }
   });
}
```

.with() fetches relational DTO properties, nesting permitted (“a.b”, “a.b.c.d”)

RequestFactory Goodies: Validation

- Annotate entities or value types per JSR303
  - @NotNull, @Length(min=0,max=5), ...
- Prior to invoking any service method on the server, RequestFactory calls validation-api on entity and value types
- Returns Set<ConstraintViolation> to client
- Handle in Receiver's onViolation() method

RequestFactory Goodies: Editor Framework

- without Editor, must sync DTO <==> UI
  - nameLabel.setValue(person.getName());
  - person.setName(nameLabel.getValue());
- with Editor (see DynaTableRF sample)
  - PersonView implements PersonEditor<PersonProxy>
  - PersonEditor has ValueBoxEditor<String> name, ...
  - RequestFactoryEditorDriver<PersonProxy, PersonEditor>
    - edit(), save(), flush()
- Editor automatically gives you the property list for .with()

Putting it together: RequestFactory + Objectify

- Annotate entities with Objectify
- Define *EntityProxy* interfaces
- Expose DAOs directly as RequestFactory services!
- Consider a CRUD service with 4 methods:

GWT-RPC Command pattern	RequestFactory
4 Actions + 4 Results	1 *RequestContext*
4 ActionHandlers	1 *EntityProxy*
1 DAO	1 DAO
1 Entity	1 Entity
14 classes	**2 classes, 2 interfaces**

DAO as RequestFactory Service

com.myapp.server.NamedListDao

```
public class NamedListDao extends ObjectifyDao<NamedList>
{
   public void save(NamedList list)
   {
      AppUser loggedInUser = LoginService.getLoggedInUser();
      list.setOwner(loggedInUser);
      this.put(list);
   }
    ...
}
```

com.myapp.shared.NamedListService

```
@Service(value = NamedListDao.class, locator = DaoServiceLocator.class)
public interface NamedListService extends RequestContext {
  Request<Void> save(NamedListProxy list);
  ...
}
```

How does it work?

- Entity Locators
  - Entities must implement static finder methods or delegate to a Locator
  - Can use one ObjectifyLocator for all entities
- ServiceLocators
  - Enables services to be configured through dependency injection (Spring, Guice, ...)
  - Instance services require ServiceLocator
  - Can use a generic ServiceLocator for all services

Objectify Entity Locator

```
public class ObjectifyLocator extends Locator<DatastoreObject, Long> {
  @Override
  public DatastoreObject create(Class<? extends DatastoreObject> clazz) {
      ...
      return clazz.newInstance();
      ...
  }

  @Override
  public DatastoreObject find(Class<? extends DatastoreObject> clazz, Long id) {
    DAOBase daoBase = new DAOBase();
    return daoBase.ofy().find(clazz, id);
  }
  ...
}
```

Summing it all up: RequestFactory + Objectify

- 1 entity Locator
- 1 ServiceLocator
- 1 RequestFactory
- Per entity
  - 1 domain class (server)
  - 1 proxy interface (shared)
- Per service
  - 1 impl (on server, may extend base class like ObjectifyDAO)
  - 1 service interface (RequestContext)
- Is there tooling for this?
  - Watch this space (Google Plugin for Eclipse)

gwt-platform

RequestFactory

Objectify

MVP with GWT-platform

Philippe Beaudoin

MVP with GWT-platform

Simplify development of large-scale GWT apps

- Presenter-centric Model-View-Presenter
  - Simple history support
  - Easy code splitting
  - Includes an Eclipse plugin
- Command pattern (on top of GWT-RPC)
- Other goodies

GWTP's Model-View-Presenter

- Not based on GWT's Activity/Places (Yet!)
- Annotation-based API, all in the presenter
  - @NameToken instead of Place + PlaceTokenizer
  - @Proxy instead of PlaceHistoryMapper + ActivityMapper
- Support for nested presenters
- Built-in GIN support
- Much more!

Writing a basic View

```
public class WelcomeView extends ViewImpl
    implements WelcomePresenter.MyView {

  interface Binder extends UiBinder<Widget, WelcomeView> { }
  private final Widget widget;

  @Inject
  public WelcomeView(Binder binder) {
    widget = binder.createAndBindUi(this);
  }

  @Override
  public Widget asWidget() {
    return widget;
  }
}
```

Writing a basic Presenter

```
public class WelcomePresenter extends
    Presenter<WelcomePresenter.MyView, WelcomePresenter.MyProxy> {

  public interface MyView extends View {}

  @ProxyCodeSplit
  public interface MyProxy extends ProxyPlace<MainPagePresenter> {}

  @Inject
  public MainPagePresenter(EventBus eventBus, MyView view,
      MyProxy proxy) {
    super(eventBus, view, proxy);
  }

  // ...
}
```

Attaching the Presenter to the URL

```
public class WelcomePresenter extends
    Presenter<WelcomePresenter.MyView, WelcomePresenter.MyProxy> {

  public interface MyView extends View {}

  @ProxyCodeSplit
  @NameToken("WELCOME")
  public interface MyProxy extends ProxyPlace<MainPagePresenter> {}

  @Inject
  public MainPagePresenter(EventBus eventBus, MyView view,
      MyProxy proxy) {
    super(eventBus, view, proxy);
  }

  // ...
}
```

Receiving parameters via the URL

```
public class WelcomePresenter extends
    Presenter<WelcomePresenter.MyView, WelcomePresenter.MyProxy> {

  public interface MyView extends View {
    void setName(String name);
  }

  @ProxyCodeSplit
  @NameToken("WELCOME")
  public interface MyProxy extends ProxyPlace<MainPagePresenter> {}

  @Inject
  public MainPagePresenter(EventBus eventBus, MyView view,
      MyProxy proxy) {
    super(eventBus, view, proxy);
  }

  @Override
  public void prepareFromRequest( PlaceRequest request ) {
    getView().setName(request.getParameter( "name", "Anonymous" ));
    super.prepareFromRequest( request );
  }
}
```

Nested presenters

- Leverages the well-known widget hierarchy paradigm
- Maintains the clear UI/Logic separation
- Brings lifecycle events to the Presenters
  - onReset
  - onReveal
  - onHide

Nesting presenters (the View)

```
public class MainView extends ViewImpl
    implements MainPresenter.MyView {

  @UiField FlowPanel topSlot;
  @UiField FlowPanel centerSlot;

  @Override
  public void setInSlot(Object slot, Widget content) {
    if (slot == MainPresenter.TOP_SLOT) {
      topSlot.clear();
      topSlot.add(content);
    } else if (slot == MainPresenter.CENTER_SLOT) {
      centerSlot.clear();
      centerSlot.add(content);
    } else {
      super.setInSlot(slot, content);
    }
  }
```

Nesting presenters (the Presenter)

```
public class MainPresenter extends
    Presenter<MainPresenter.MyView, MainPresenter.MyProxy> {

  // ...

  public static final Object TOP_SLOT = new Object();

  @ContentSlot
  public static final Type<RevealContentHandler<?>> CENTER_SLOT
      = new Type<RevealContentHandler<?>>();

  // ...
}
```

Revealing top-level presenters

```
public class MainPresenter extends
    Presenter<MainPresenter.MyView, MainPresenter.MyProxy> {

  public static final Object TOP_SLOT = new Object();

  @ContentSlot
  public static final Type<RevealContentHandler<?>> CENTER_SLOT
      = new Type<RevealContentHandler<?>>();

  @Override
  protected void revealInParent() {
    RevealRootContentEvent.fire(this, this);
  }
}
```

- To use GWT's LayoutPanels
  - RevealRootLayoutContentEvent

Revealing nested presenters

```
public class WelcomePresenter extends
    Presenter<WelcomePresenter.MyView, WelcomePresenter.MyProxy> {

  @Override
  protected void revealInParent() {
    RevealContentEvent.fire( this,
      MainPresenter.CENTER_SLOT, this );
  }
}
```

- Event bus implies low coupling

PresenterWidgets

- Sometimes you don't want a URL
- Or you may want a "reusable presenter"
- Really, just a widget with UI/Logic separation
- Enters: PresenterWidget
  - Trades low coupling for simplicity of use

Using a PresenterWidget

```
public class MainPresenter extends
    Presenter<MainPresenter.MyView, MainPresenter.MyProxy> {

  public static final Object TOP_SLOT = new Object();
  @ContentSlot
  public static final Type<RevealContentHandler<?>> CENTER_SLOT
      = new Type<RevealContentHandler<?>>();

  private final LoginPresenter loginPresenter;
  @Inject
  public MainPagePresenter(EventBus eventBus, MyView view,
      MyProxy proxy, LoginPresenter loginPresenter) {
    super(eventBus, view, proxy);
    this.loginPresenter = loginPresenter;
  }

  @Override
  protected void onReveal() {
    super.onReveal();
    setInSlot(TOP_SLOT, loginPresenter);
  }
}
```

Connecting everything together

- Presenters need to be accessible from the ginjector
- You need to bind presenters and views. (Gin)
- Custom PlaceManager (to define a default page)
- See the wiki...
- Better yet: the GWTP Eclipse plugin does it for you!

GWTP Summary

- Two classes:
  - One Presenter, One View
- Super simple history @NameToken("DONE")
- Nesting
  - Loosely coupled or via PresenterWidget
- Explore!
  - Tabbed presenters, dialog box presenters
  - Gatekeepers (client-side security)
  - @ProxyEvent
  - Breadcrumbs
  - Annotation processors (events, actions, DTOs)
  - GWTP's command pattern

Google I/O 11

Conclusion

- Sample apps
  - http://code.google.com/p/listwidget/ (Maven, Objectify, RequestFactory)
  - http://code.google.com/p/gwtgae2011/ (Maven, Objectify, RequestFactory, gwt-platform)
- Stay tuned
  - http://turbomanage.wordpress.com
  - http://www.arcbees.com
  - @googledevtools

- Thank you to the open source community around GWT, GAE!
- Get involved

More GWT at I/O

- Android + App Engine: A Devloper's Dream Combination
  - preview of RequestFactory tooling in GPE
  - today @ 3:45
- Office hours
  - 12-3pm
  - Level 2, Alcove 3
- Visit the GWT Developer Sandbox
  - Meet companies using GWT
  - Find special offers

Thank you!

http://code.google.com

Hashtags: #io2011 #DevTools
Feedback: http://goo.gl/Rj0NA