POJO Entry Points

Related pages:

- Custom Match Rules

This topic describes how to create custom match rules for POJO (Plain Old Java Object) applications. See Custom Match Rule Live Preview for instructions about interactively working with live data to create a POJO.

About POJO Custom Match Rules

Custom match rules for POJOs let you configure business transaction detection in application environments that run pure Java applications or that use frameworks with entry points that are not automatically detected.

Unlike common frameworks, which are characterized by well-known entry points for applications, the logical entry point for a business transaction for a POJO entry point could be any method in the application.

To configure a custom POJO entry point, therefore, you need to identify the method that AppDynamics should consider the business transaction entry point. Keep in mind that the start and end of the execution of the method will correspond to the start and end of the business transaction, so the method should encapsulate the complete execution of the business transaction.

For example, consider the method execution sequence:

```java
com.foo.threadpool.WorkerThread.run()
calls com.foo.threadpool.WorkerThread.runInternal()
calls com.foo.Job.run()
```

The first two calls to `run()` method are the blocking methods that accept a job and invoke it. The `Job.run()` method is the actual unit of work, because Job is executed every time the business transaction is invoked and finishes when the business transaction finishes.
Methods like these are the best candidates for POJO entry points. The response time for POJO transactions is measured from this entry point, and remote calls are tracked the same way as are remote calls for a Servlet's Service method.

Creating POJO Custom Match Rules

You can create a POJO entry point by adding a custom match rule with `POJO` as the Entry Point Type. The agent names business transactions for the custom match rule.

In order for the agent to discover POJO transactions, your custom match rule must belong to a scope that also includes a Default Java Automatic Transaction Discovery rule with POJO transaction monitoring enabled.

You can classify matching transactions as background tasks by enabling the Background Task check box. When a request runs as a background task, AppDynamics reports only Business Transaction metrics for the request. It does not aggregate response time and calls metrics at the tier and application levels for background tasks. This ensures that background tasks do not distort the baselines for the business application. Also, you can set a separate set of thresholds for background tasks. For more information, see Monitor Background Tasks.

The custom match rule configuration offers several options for matching method invocations and splitting matched calls into separate business transactions, as described below.

If you are running on IBM JVM v1.5 or v1.6, you must restart the JVM after defining the custom match rules.

Matching by Class and Method

You can specify matching criteria for the custom rule based on various forms of a method or class to which the method belongs.

When specifying the method name matching criteria, use parameter matching to match against a particular method signature. For example, say you want to instrument one of more methods in the following class:

```java
class A {
    public void m1();
    public void m1(String a);
    public void m1(String a, com.mycompany.MyObject b);
}
```

Configure instrumentation for each method based on its signature as follows:

- To instrument the first method signature (with no parameters), create a POJO-based business transaction match rule as follows:
  - Match Classes: with a Class Name that Equals A
  - Method Name: Equals m1()

- To instrument the second method, create a match rule as follows:
  - Match Classes: with a Class Name that Equals A
  - Method Name: Equals m1(java.lang.String)

- To instrument the third method, create a match rule as follows:
  - Match Classes: with a Class Name that Equals A
  - Method Name: Equals m1(java.lang.String, com.mycompany.MyObject)
You can also match methods that belong to classes with certain annotations. For example, say you want to match all classes that are annotated with `@com.acme.Processor`. The custom rule configuration can define the annotation as follows:

```
@com.acme.Processor
```

When calls to the `process()` method match, the result is a business transaction named for the custom match rule name.

### Matching on Inner Classes and Inner Interfaces

You can match on inner classes and inner interfaces by adding a `$` sign after the class name. For example, you could specify the following for the class name:

```
com.my.package.OuterClass$InnerClass
```

### Splitting Matches into Separate Business Transactions

With only transaction matching configured, all requests matched by the match rule belong to the custom business transaction. Alternatively, you can create a match rule that generates multiple named business transaction based on criteria in the request.

For example, the following configuration defines a custom match rule that matches by superclass. In the example, the entry point matches the `process()` method defined in classes that extend the `com.acme.AbstractProcessor` superclass. In our example, the superclass is extended by the subclasses `SalesProcessor`, `InventoryProcessor`, and `BacklogProcessor`.

1. Create a custom rule that matches the superclass matches all of those subclasses:
2. Split the transaction on class name, you can have separate business transactions created for the respective subclasses. AppDynamics names the transactions with the class name, prepending the match rule name to the class name. For example, `Process.SalesProcessor`, `Process.InventoryProcessor`, and `Process.BacklogProcessor`.
It's possible that parameter values carry what would be meaningful identifiers for business transactions in your application. For example, say the jobType parameter in the following method may have the values of Sales, Inventory or Backlog.

```java
public void process(String jobType, String otherParameters...)
```

You can split the transaction based on parameter value by indicating the zero-based position of the parameter in the method signature, 0 in the example:
The `toString()` method indicates how the value of the parameter should be rendered. See Using Getter Chains for more information. As indicated in the dialog, you can use other transaction splitting criteria as well, including thread ID, method name, class name, and so on.

The name of the rule is prepended to the dynamically-generated name to form the business transaction name.

**Excluding Matches from Transaction Splitting**

An exclude rule defines criteria which, when matched by a transaction instance, prevents transaction splitting from occurring as otherwise configured. In effect, this nullifies the effects of transaction splitting for the subset of transactions matched by the exclude condition.