

# Применение статического анализа типов потоков для разграничения доступа в многопоточных приложениях на Java

Выполнил:

И.В. Егоров

Научный

руководитель:

А.А. Бреслав

Рецензент:

В.С. Полозов

# Анализ многопоточных программ

- Средства поиска и предотвращения ошибок
  - Исключение взаимодействия потоков
  - Введение иерархий владения, мониторы
  - Динамические средства анализа
- Предлагаемый подход
  - Статическое разделение потоков по типам в зависимости от исполняемых функций

# Пример применения

```
class ProductFactory {  
    @ExecPermissions(Consumer.class)  
    public Product consume() {  
        ...  
    }  
    @ExecPermissions(Producer.class)  
    public void produce() {  
        ...  
    }  
}
```

```
@ThreadMarker  
interface Producer {}
```

```
@ThreadMarker  
interface Consumer {}
```

```
class ProducerTask implements  
    Runnable, Producer {  
  
    public ProducerTask(ProductFactory  
        factory, int count) {  
        this.factory = factory;  
        this.count = count;  
    }  
  
    public void run() {  
        for (int i = 0; i < count; ++i) {  
            factory.produce();  
        }  
    }  
}
```

# Пример применения

```
class ProductFactory {  
    @ExecPermissions(Consumer.class)  
    public Product consume() {  
        ...  
    }  
    @ExecPermissions(Producer.class)  
    public void produce() {  
        ...  
    }  
}
```

```
@ThreadMarker  
interface Producer {}
```

```
@ThreadMarker  
interface Consumer {}
```

```
class ProducerTask implements  
    Runnable, Producer {  
  
    public ProducerTask(ProductFactory  
        factory, int count) {  
        this.factory = factory;  
        this.count = count;  
    }  
  
    public void run() {  
        for (int i = 0; i < count; ++i) {  
            factory.produce();  
            factory.consume();  
        }  
    }  
}
```

# Модель

- Модель ограничений доступа =>  
модель классов потоков
- Ограничения доступа задаются посредством аннотаций

Аннотация	Параметры	Область применения
@ThreadMarker	-	Интерфейс
@ExecPermissions	классы потоков	Метод
@ReadPermissions		Поле класса
@WritePermissions		
@ThreadStarter	-	Аргумент функции или поле класса

# Реализация прототипа

The screenshot shows the Eclipse IDE with two Java files open: `ProductFactory.java` and `ProducerTask.java`.

**ProductFactory.java**

```
package demo;

import com.google.code.annatasha.annotations.ThreadMarker;
import com.google.code.annatasha.annotations.Method.ExecPermissions;

public final class ProductFactory {

    @ExecPermissions(Producer.class)
    public void produce() {

    }

    @ExecPermissions(Consumer.class)
    public Object consume() {
        return new Object();
    }

}

@ThreadMarker
interface Producer {}

@ThreadMarker
interface Consumer {}
```

**ProducerTask.java**

```
package demo;

public class ProducerTask implements Runnable, Producer {

    @Override
    public void run() {
        ProductFactory factory = new ProductFactory();
        factory.produce();
        factory.consume();
    }

}
```

The `factory.consume();` line in `ProducerTask.java` is highlighted in blue, and a red 'x' icon is visible in the left margin next to it, indicating an error.

**Problems View**

1 error, 0 warnings, 0 others

Description	Resource	Path	Type
▼ Errors (1 item)			
Method attempts to execute inaccessible method	ProducerTask.java	demo/src/demo	Annatasha Problem

Method attempts to execute inaccessible method

# Результаты

- Модель классов потоков
- Модель модификаторов доступа
- Доказано:  
    модель модификаторов доступа =>  
        модель классов потоков
- Прототип инструментального средства
- Прототип опробован на примере
  - Параллельные вычисления с матрицами