Struts2 是一个流行的 Java Web 应用框架，它通过拦截器（Interceptor）机制来处理请求。拦截器在请求处理过程中扮演了重要角色，可以在请求到达 Action 之前和响应返回客户端之前执行特定的逻辑。以下是关于 Struts2 拦截器工作原理、内建拦截器以及自定义拦截器方法的简述：

Struts2 拦截器的工作原理

1. **请求进入**: 当客户端发出请求时，Struts2 前端控制器（DispatcherServlet）接收请求。
2. **查找配置**: 前端控制器根据 struts.xml 或注解查找与请求相对应的 Action 配置。
3. **拦截器链**: Action 配置中定义的拦截器链将被应用。拦截器链是一个拦截器的列表，这些拦截器按顺序执行。
4. **执行拦截器**:
   - **前置处理**: 每个拦截器先执行 `intercept` 方法中的前置处理逻辑。
   - **执行 Action**: 前置处理完成后，请求会被传递到下一个拦截器，直到最终到达 Action。
   - **后置处理**: Action 执行完成后，拦截器的后置处理逻辑被执行（如果有）。
5. **生成响应**: Action 返回结果，拦截器链按相反顺序执行后置处理，最终将响应返回给客户端。

Struts2 内建的拦截器

Struts2 提供了许多内建拦截器，常见的包括：

1. **params**: 处理 HTTP 请求参数，将其注入到 Action 的字段中。
2. **validation**: 进行 Action 验证，执行 validate 方法或配置的验证文件。
3. **workflow**: 处理工作流逻辑，常用于在验证失败时返回输入页面。
4. **conversionError**: 处理类型转换错误，将错误信息放入 Action 的 fieldErrors 中。
5. **logger**: 记录请求处理过程中的日志信息。
6. **exception**: 处理 Action 中抛出的异常，提供一个统一的异常处理机制。

自定义拦截器的三种方法

1. **实现 Interceptor 接口**:
   - 创建一个类实现 `com.opensymphony.xwork2.interceptor.Interceptor` 接口。
   - 实现 `init()`, `destroy()`, `intercept(ActionInvocation invocation)` 方法。
   - 在 `intercept` 方法中添加自定义逻辑，并调用 `invocation.invoke()` 传递控制权。

```
public class MyInterceptor implements Interceptor {
```

```
    public void init() {}

    public void destroy() {}

    public String intercept(ActionInvocation invocation) throws Exception {

        // 前置处理逻辑

        String result = invocation.invoke(); // 传递到下一个拦截器或Action

        // 后置处理逻辑

        return result;

    }

}
```

2. **继承 AbstractInterceptor**:
   - 继承 `com.opensymphony.xwork2.interceptor.AbstractInterceptor` 类。
   - 重写 `intercept` 方法即可。

```
public class MyAbstractInterceptor extends AbstractInterceptor {

    @Override

    public String intercept(ActionInvocation invocation) throws Exception {

        // 前置处理逻辑

        String result = invocation.invoke(); // 传递到下一个拦截器或Action

        // 后置处理逻辑

        return result;

    }

}
```

3. **通过注解配置**:
   - 使用 `@Interceptor` 注解来定义拦截器。
   - 然后在 Action 类上使用 `@InterceptorRef` 注解来引用这个自定义拦截器。

```
@Interceptor
public class MyAnnotatedInterceptor extends AbstractInterceptor {
    @Override
    public String intercept(ActionInvocation invocation) throws Exception {
        // 前置处理逻辑
        String result = invocation.invoke(); // 传递到下一个拦截器或Action
        // 后置处理逻辑
        return result;
    }
}

@Action(value="myAction", interceptorRefs={
    @InterceptorRef("myAnnotatedInterceptor")
})
public class MyAction extends ActionSupport {
    public String execute() {
        return SUCCESS;
    }
}
```

通过这些方法，可以灵活地在 Struts2 应用中实现各种功能的拦截器，满足不同的业务需求。

2.（高1）

import java.util.ArrayList;

import java.util.List;

class Record {

```
private static int counter = 0;

private int id;

private String data;
```

```
public Record(String data) {

    this.id = counter++;

    this.data = data;

}
```

```
public int getId() {

    return id;

}
```

```
public String getData() {

    return data;

}
```

}

class RecordStore {

```
private List<Record> records;
```

```
public RecordStore() {

    records = new ArrayList<>();

}
```

```
public void addRecord(String data) {

    records.add(new Record(data));

}
```

```
public void deleteRecord(int index) {

    if (index >= 0 && index < records.size()) {

        records.remove(index);

    } else {

        System.out.println("Invalid index to delete");

    }

}
```

```
public void printRecords() {

    for (Record record : records) {

        System.out.println("ID: " + record.getId() + ", Data: " +
record.getData());

    }

}
```

}

public class Main {

```
public static void main(String[] args) {

    RecordStore store = new RecordStore();
```

```
    // 添加三条数据[A, B, C]
    store.addRecord("A");
    store.addRecord("B");
    store.addRecord("C");
```

```
    System.out.println("Initial records:");
    store.printRecords();
```

```
    // 删除第二条数据
    store.deleteRecord(1);
```

```
    System.out.println("\nAfter deleting second record:");
    store.printRecords();
```

```
    // 再添加一条数据
    store.addRecord("D");
```

```
    System.out.println("\nAfter adding a new record:");
    store.printRecords();
 }
```

```
}
```

3（高2）

```
import javax.microedition.midlet.*;
import javax.microedition.rms.*;

public class RecordStoreExample extends MIDlet {
```

```
private RecordStore recordStore;
```

```
public void startApp() {

    try {

        // 打开或创建一个名为 "MyRecordStore" 的 RecordStore

        recordStore = RecordStore.openRecordStore("MyRecordStore", true);
```

```
        // 添加三条数据[A, B, C]

        addRecord("A");

        addRecord("B");

        addRecord("C");
```

```
        // 输出初始记录

        System.out.println("Initial records:");

        printRecords();
```

```
        // 删除第二条数据

        deleteRecord(2);
```

```
        // 输出删除后的记录

        System.out.println("\nAfter deleting second record:");

        printRecords();
```

```
        // 再添加一条数据

        addRecord("D");
```

```
        // 输出最终记录
        System.out.println("\nAfter adding a new record:");
        printRecords();
```

```
        // 关闭 RecordStore
        recordStore.closeRecordStore();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
public void pauseApp() {
    // 暂停应用
}
```

```
public void destroyApp(boolean unconditional) {
    // 销毁应用，关闭 RecordStore
    try {
        if (recordStore != null) {
            recordStore.closeRecordStore();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
private void addRecord(String data) throws RecordStoreException {

    byte[] recordData = data.getBytes();

    recordStore.addRecord(recordData, 0, recordData.length);

}
```

```
private void deleteRecord(int recordId) throws RecordStoreException {

    recordStore.deleteRecord(recordId);

}
```

```
private void printRecords() throws RecordStoreException {

    RecordEnumeration enumeration = recordStore.enumerateRecords(null, null,
false);

    while (enumeration.hasNextElement()) {

        int id = enumeration.nextRecordId();

        byte[] recordData = recordStore.getRecord(id);

        String data = new String(recordData);

        System.out.println("ID: " + id + ", Data: " + data);

    }

}
```

}